

ON THE EQUIVALENCE OF MARKOV ALGORITHMS AND TURING MACHINES AND SOME CONSEQUENT RESULTS

Eleftherios Papathanassiou

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews



1979

Full metadata for this item is available in
St Andrews Research Repository
at:

<http://research-repository.st-andrews.ac.uk/>

Please use this identifier to cite or link to this item:

<http://hdl.handle.net/10023/13736>

This item is protected by original copyright

ABSTRACT.

Turing Machines and Markov Algorithms are, and were designed to be, the most powerful devices possible in the field of abstract automata: by their means any and every computable function can be computed.

Because of their equal, indeed maximal, strength, it was naturally assumed that these devices should be equivalent. Nonetheless a formal, exact proof of this universally presumed equivalence was lacking.

The present dissertation rectifies that omission by developing the desired complete, rigorous proof of the equivalence between Turing Machines and Markov Algorithms. The demonstration is being conducted in a constructionist way: for any given Markov Algorithm it is shown that a Turing Machine can be constructed capable of performing exactly what the Algorithm can do and nothing more, and vice versa.

The proof consists in the theoretical construction, given an arbitrary Markov Algorithm, of a Turing Machine behaving in exactly the same way as the Algorithm for all possible inputs; and conversely. Furthermore, the proof is given concrete shape by designing a computer program which can actually carry out the said theoretical constructions.

The equivalence between TM and MA as proven in the first part of our thesis, is being used in the second part for establishing some important consequent results: Thus the equivalence of Deterministic and Nondeterministic MA, of TM and

ProQuest Number: 10167348

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10167348

Published by ProQuest LLC (2017). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Type 0 Grammars, and of Labelled and Unlabelled MA is concisely shown, and the use of TM as recognizers for type 1 and 3 grammars exclusively is exhibited. It is interesting that, by utilizing the equivalence of TM and MA, it was made possible that the proofs of these latter results be based on primitive principles.

I hereby declare that this thesis has been composed by myself; that the work of which it is a record has been done by myself; and, that it has not been accepted in any previous application for any higher degree. This research concerning the equivalence between Markov Algorithms and Turing Machines was undertaken on 1st October 1974, the date of my admission as a research student under Ordinance General No.12 for the degree of Doctor of Philosophy (Ph.D).

Eleftherios A. Papathanassiou

I hereby declare that the conditions of the Ordinance and Regulations for the degree of Doctor of Philosophy (Ph.D) at the University of St Andrews have been fulfilled by the candidate, Eleftherios A. Papathanassiou.

Professor A. J. Cole

ON THE EQUIVALENCE OF MARKOV ALGORITHMS AND
TURING MACHINES AND SOME CONSEQUENT RESULTS

ON THE EQUIVALENCE OF MARKOV ALGORITHMS AND
TURING MACHINES AND SOME CONSEQUENT RESULTS

by

Eleftherios A. Papathanassiou

A thesis submitted for the degree of Doctor of Philosophy

Department of Computational Science,
University of St Andrews
St Andrews, Scotland



May 1979

ΑΦΙΕΡΟΥΤΑΙ
ΕΙΣ ΤΟΥΣ ΓΟΝΕΙΣ ΜΟΥ

(Dedicated to my parents)



PREFACE

Turing Machines and Markov Algorithms are, and were designed to be, the most powerful devices possible in the field of abstract automata: by their means any and every computable function can be computed.

Because of their equal, indeed maximal, strength, it was naturally assumed that these devices should be equivalent. Nonetheless a formal, exact proof of this universally presumed equivalence was lacking.

The present dissertation rectifies that omission by developing the desired complete, rigorous proof of the equivalence between Turing Machines and Markov Algorithms. The demonstration is being conducted in a constructionist way: for any given Markov Algorithm it is shown that a Turing Machine can be constructed capable of performing exactly what the Algorithm can do and nothing more, and vice versa.

The proof consists in the theoretical construction, given an arbitrary Markov Algorithm, of a Turing Machine behaving in exactly the same way as the Algorithm for all possible inputs; and conversely. Furthermore, the proof is given concrete shape by designing a computer program which can actually carry out the said theoretical constructions.

The equivalence between TM and MA as proven in the first part of our thesis, is being used in the second part for establishing some important consequent results: Thus the equivalence of Deterministic and Nondeterministic MA, of TM and Type 0 Grammars,

and of Labelled and Unlabelled MA is concisely shown, and the use of TM as recognizers for type 1 and 3 grammars exclusively is exhibited. It is interesting that, by utilizing the equivalence of TM and MA, it was made possible that the proofs of these latter results be based on primitive principles.

In pursuing the research leading to this dissertation I was fortunate enough to enjoy the benefit of the constant help and continuous advice on the part of my supervisor, Professor A.J. Cole, whom it is my pleasurable duty here to thank deeply for all that he did towards the completion of my work in matters both academic and more practical, though no less necessary.

I would also like to express my thanks to the Ministry of Coordination of Greece, Directory of Technical Aid, for making available the funds enabling me to continue and complete my thesis.

In the process of my work I was able to freely use the facilities available in the Department of Computational Science, I am grateful for this.

As to the typing of the thesis, I cannot pass unnoticed the care and perseverance bestowed upon a formidable job by Mrs C.G. MacArthur. My thanks are due to her.

CONTENTS

INTRODUCTION

1. ALPHABETS, STRINGS AND SYMBOLS	2.
2. LANGUAGES	2.
3. MARKOV ALGORITHMS	3.
4. TURING MACHINES	5.
5. GRAMMARS	7.
5.1 Types of Grammars	9.
6. GRAMMATICAL DEVICES	9.

CHAPTER I

FOR EVERY MARKOV ALGORITHM THERE IS AN EQUIVALENT
TURING MACHINE

1. INTRODUCTION	12.
2. DEFINITION OF TM	13.
3. AN OUTLINE OF THE CONSTRUCTION	14.
4. CONSTRUCTION OF TM	
4.1 Introduction	15.
4.2 First TM move included	17.
4.3 Initial and Final TM moves for every MA rule	17.
4.4 Meaning of subscript "w"	18.
4.5 Meaning of θ	19.
4.6 Applicability search for rule i	20.
4.7 Last character of LHS	23.
4.8 Substitution	25.
4.8.1 Case A: substitution for $r=s$	25.
4.8.2 Case B: substitution for $r<s$	27.
4.8.2.1 Explanation of D_p	28.

4.3	Routine "Expansion"	29.
4.8.3.1	Moves of the routine "Expansion" included	31.
4.8.4	Substitution following the routine expansion	34.
4.8.5	Case C $r > s$	35.
4.8.5.1	The unwanted squares "are transferred" to the left of the first blank symbol "B"	36.
4.8.5.2	Pseudosymbols	36.
4.8.5.3	The unwanted squares are transferred to the left of T_i	38.
4.8.6	Case C $r > s$	38.
4.8.7	Case C, Moves	40.
4.8.8	Description and moves of "Cut off"	41.
4.8.8.1	An example of the tape after the "Cut off" execution	43.
4.8.9	First moves after the termination of "Cut off"	43.
4.9	Special case $ LHS_i = 0$	44.
4.10	Values of subscript w	47.
5.	SOME IMPORTANT REMARKS FOR THE TM's CONSTRUCTION	48.
6.	TERMINATION OF TM	50.
	THEOREM 1.1	52.
7.	EQUIVALENCE BETWEEN AN MA RULE AND A TM SET OF MOVES	65.
	THEOREM 1.2	66.
	THEOREM 1.3	69.
8.	TWO COMMENTS	70.
	THEOREM 1.4	71.
	THEOREM 1.5	75.
9.	CONCLUSION	78.

CHAPTER II

FOR EVERY TURING MACHINE THERE IS AN EQUIVALENT
MARKOV ALGORITHM

1.	DEFINITIONS OF TM AND MA	81.
2.	SIMULATION OF TM'S STATES AND HEAD-MOVEMENTS BY MA	81.
3.	CORRESPONDING MA RULES TO TM MOVES	82.
4.	CONSTRUCTION OF MA	83.
5.	A SYMBOLIZATION FOR TM	85.
	THEOREM 2.1	86.
6.	ACCEPTANCE AND REJECTION OF A GIVEN STRING BY TM AND MA	93.
	THEOREM 2.2	94.
	THEOREM 2.3	98.
7.	A LABELLED MARKOV ALGORITHM SIMULATING TM	102.
8.	CONCLUSION	106.

CHAPTER III

NONDETERMINISM IN MARKOV ALGORITHMS AND TURING MACHINES

1.	INTRODUCTION	108.
2.	NONDETERMINISTIC GRAMMATICAL DEVICES	108.
3.	A NONDETERMINISTIC TURING MACHINE	109.
4.	NONDETERMINISTIC MARKOV ALGORITHMS	110.
	4.1 Cases of "more or less Nondeterministic" Markov Algorithms	112.
5.	A NONDETERMINISTIC TURING MACHINE FOR A GNMA	114.
	5.1 Definition of sets δ_u and δ_a	115.
	5.2 Equivalence of NTM and GNMA	117.
6.	A NONDETERMINISTIC MARKOV ALGORITHM FOR A NON- DETERMINISTIC TURING MACHINE	118.
	6.1 Differences in the corresponding constructions	119.

6.2	The constructed GNMA	122.
6.3	Equivalence of GNMA and NTM	123.
7.	EQUIVALENCE BETWEEN NONDETERMINISTIC AND DETERMINISTIC MARKOV ALGORITHMS	123.

CHAPTER IV

CONSEQUENT RESULTS OF THE EQUIVALENCE

1.	INTRODUCTION	126.
2.	EQUIVALENCE OF LABELLED AND UNLABELLED MARKOV ALGORITHMS	126.
3.	TURING MACHINES AND TYPE 0 GRAMMARS	127.
4.	NONDETERMINISTIC AND DETERMINISTIC TYPE 0 GRAMMARS	129.
5.	TYPE 0 GRAMMARS AND TYPE 0 PRODUCTIONS	130.
6.	TURING MACHINES AND TYPE 3 GRAMMARS	133.
6.1	Turing Machines recognizing Type 3 Grammars	134.
6.2	A Turing Machine simulating a type 3 Recognizing Grammar	136.
6.3	Conclusion	137.
7.	TURING MACHINES AND TYPE 1 GRAMMARS	138.
7.1	Turing Machines Recognizing Type 1 Grammars	139.

CHAPTER V

COMPUTER PROGRAMS FOR DEMONSTRATING THE EQUIVALENCE OF TURING MACHINES AND MARKOV ALGORITHMS

1.	INTRODUCTION	142.
2.	A TURING MACHINE SIMULATOR	142.
2.1	Representation of a Turing Machine in a Computer	142.
2.2	Input-output	143.
2.3	Subroutines Used	144.
2.4	A Version with double-subscript states	144.

3.	A MARKOV ALGORITHM SIMULATOR	145.
3.1	Input to the Program	145.
3.2	Data Structure	146.
3.3	Procedures used	149.
3.3.1	Procedure Apply	151.
3.4	Subscript Version	152.
4.	A PROGRAM PRODUCING A TURING MACHINE EQUIVALENT TO A GIVEN MARKOV ALGORITHM	152.
4.1	The Problem of Generics	152.
4.2	Data Structure	159.
4.3	The Program	163.
4.4	Generics, next states and next moves	164.
4.5	Procedure "Leftempty"	167
4.6	Procedure "Prepare"	168.
4.7	Procedure "Filltable"	169.
4.8	Procedure "Firstmoves"	172.
4.9	Procedure "Firstsym"	173.
4.10	Procedure "Symbexp"	173.
4.11	Procedure "Firstgen"	174.
4.12	Procedure "Nextmoves"	175.
4.13	Procedure "Simplesym"	175.
4.14	Procedure "Usedgen"	175.
4.15	Procedure "Unusedgen"	176.
4.16	Procedure "Caseone"	176.
4.17	Procedure "Casethree"	177.
4.18	Procedure "Casetwo"	177.
4.19	Procedure "Expand"	178.
4.20	Procedure "Contract"	178.
4.21	Procedure "Put", "Nextrule"	179.
5.	A PROGRAM PRODUCING A MARKOV ALGORITHM EQUIVALENT TO A GIVEN TURING MACHINE	179.

5.1 Data Structure	180.
5.2 Procedures used	181.
BIBLIOGRAPHY	183.

INTRODUCTION

1. ALPHABETS, STRINGS AND SYMBOLS

We will use the name Alphabet for a finite set of characters (or symbols). Thus if Σ is an alphabet, a string (or word, or sentence) over Σ is a finite sequence of symbols of Σ .

With Λ or ϵ we symbolize the empty string, that is the string composed of no symbols.

The set of all strings over Σ is denoted by Σ^* ; and the set $\Sigma^* - \{\epsilon\}$ is denoted by Σ^+ .

The concatenation of two strings τ and x is denoted by τx . For a string τ and the empty string ϵ we have:

$$\epsilon \tau = \tau \epsilon = \tau.$$

The length of a string τ , that is the number of characters in τ , will be symbolized by $|\tau|$. If $\tau = \epsilon$, obviously $|\tau| = 0$.

2. LANGUAGES

Let us suppose that Σ is a finite alphabet and L is a set of strings over Σ . That is $L \subseteq \Sigma^*$. The strings in L may have been chosen either arbitrarily, or because the formation of some or all of them may obey some kind of common "laws". Possibly all or some of these "laws" may be obeyed by all strings in L or by sets of strings constituting in this way subsets of L .

Any subset of Σ^* defined as the above mentioned set L , will be called a Language over the alphabet Σ .

Since every Language is a subset of an infinite set (in our definition $L \subseteq \Sigma^*$), it may be of finite or infinite type. It is obvious that every Language which has been constructed by arbitrarily choosing its strings, one by one, can only be of finite type. We cannot of course say the same for Languages which are constituted

by strings or set of strings obeying the same "laws". This latter kind of Languages may be of finite or infinite type.

A useful "tool" for the latter kind of Languages would be a device able to "generate" or to "recognize" only those strings which belong to a given Language. Two general kinds of devices for this purpose are the various types of Markov Algorithms and the various types of Automata.

3. MARKOV ALGORITHMS

A Markov Algorithm M_a is a system:

$$M_a = (A, \Sigma, P, N_p);$$

where:

$\Sigma \subseteq A$ is a finite set of input symbols.

A is the Alphabet of M_a , that is the finite set of all symbols used by M_a .

P is a finite set of rules (or productions) of the types: " $a \rightarrow b$ " or " $a \rightarrow \cdot b$ "; where the first is a non terminating rule and the second a terminating rule, with $a, b \in A^*$. The Left Hand Side of a rule (that is the string to the left of " \rightarrow ", will be symbolized by LHS while the Right Hand Side (the string to the right of " \rightarrow ") will be symbolized by RHS. A rule is applicable when there is an occurrence of its LHS in the string w on which M_a is "working". We apply a rule by substituting an occurrence of its LHS in the string by its RHS.

N_p is a finite set of instructions for the next rule tested. (A rule is tested, by examining, (that is by searching), w to see whether or not it is applicable).

Every Markov Algorithm, for a given input string, can terminate or loop for ever. The former means that the input string was accepted and the latter that it was rejected by the Algorithm.¹

By introducing several restrictions in the above defined Markov Algorithm we can create restricted types of Markov Algorithms. Some of these restricted types of Markov Algorithms which are able to recognize or generate different types of Languages are called "Grammars".

The most common type of Markov Algorithms are the Normal Markov Algorithms defined below.

A Normal Markov Algorithm M_n is an Algorithm defined as M_a , but with the following restrictions.

1. The set P is an ordered set.
2. The applicability search, starts with rule 0, and at the beginning of the string.
3. If the last rule applied was a terminating one, the Algorithm terminates; otherwise the applicability search continues with rule 0.

1 The words "accepted" and "rejected" do not mean that the Algorithm is (in this case) exclusively a recognizing device. In fact it could equally well be a generative device. In this latter case, a string consisting of one symbol only, should be the input to the Algorithm; this symbol is called the head of the language.

- 4.. If a rule was tested and found inapplicable, the next rule in P will be tested. If however the tested rule was the last one in P the Algorithm terminates.

4. TURING MACHINES

The most powerful type of all automata are Turing Machines. It will be shown that Turing Machines can simulate the other types of automata and can generate and recognize all types of Grammars.

A Turing Machine T_m is defined as a system:

$$T_m = (\bar{K}, \Sigma, \Gamma, \delta, q_0, F)$$

where:

\bar{K}, Σ, Γ are finite sets of TM states, input symbols and all symbols used by TM respectively.

δ is a function from $\bar{K} \times \Gamma \rightarrow \bar{K} \times (\Gamma - \{B\}) \times \{L, R\}$ ¹ where B stands for the blank symbol.

q_0 is TM's initial state

$F \subseteq \bar{K}$ is the set of TM's final states.

A Turing Machine has a tape which may be infinite in both directions or in one alone (usually to the right). This tape can be considered as divided in squares (or cells) which can bear the minimum amount of information (that is a single character), at a time. The machine has also a head which can read a character from, or type one on a square of the tape when required.

Every action of the machine is determined according to the pair of its present state and the character it is reading. This action consists of the transfer of the machine to another state (possibly the same), by the printing of a symbol (possibly the same

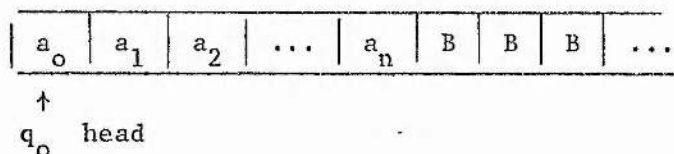
1 The function δ need not be defined for every pair in $\bar{K} \times \Gamma$ and is an "onto" function.

as the one which was read) on the scanned square and by a move of the head to the next square to the right or to the left. The whole of the described action is called a "move" and is symbolized as follows:

$$\delta\{q_i, a\} = \{q_j, b, d\}$$

where $q_i, q_j \in \bar{K}$ with $q_i \notin F$; ² $a, b \in \Gamma$; $d \in \{L, R\}$.

An input, to the machine is a string placed on the tape so that it occupies a number of successive squares equal to its length. If the tape is infinite only to the right, the string is placed on its left most squares. All the other squares of the tape are left empty (blank); we use the character B, which can only be read but not typed, to indicate that a square is blank. Thus the form of the tape, (after the input string has been placed on it) will be:



At this time the head will point to (i.e. will be scanning) the first square of the tape (i.e. the left most) and the machine will be in its initial state q_0 .

For an input string, the machine may halt or loop. The former case is possible if the machine:

- a. can ever reach a final state;
- b. can "jump" out of the tape (i.e. to the left of the left most square);
- c. has no next move defined for the pair of its most recent state and last character read;

2 If the machine is in a final state, it does not have a next move, that is it halts.

The latter case (that is the looping one) is possible if for the input string there is always defined a next move of the machine.

The machine will accept an input string, if and only if after a finite number of executions of its moves (possibly zero) it halts in one of its final states. In any other case a string will be rejected.

5. GRAMMARS

We have already mentioned that Grammars are restricted types of Markov Algorithms, (this is explained in more detail in chapter IV). In this section we will examine and define the several types of Grammars as recognizing or generating devices for their corresponding types of Languages.

Firstly we define as a Grammar G a system $G = (V_N, V_T, P, S)$ where:

V_N is a finite set of symbols called variables or nonterminals.

V_T is a finite set of symbols called terminals. For V_N and V_T we have $V_N \cap V_T = \emptyset$ and $V_N \cup V_T = V$.

P is a set of productions of the type $a \rightarrow b$, where $a \in V^+$ and $b \in V^*$ if the Grammar is generative and $a \in V^*$, $b \in V^+$ if it is recognizing.

S is the start symbol if the Grammar is Generative and ending symbol if the Grammar is Recognizing.¹

1 Since a Grammar is a type of Markov Algorithm and this is its only restriction (or instruction) for the applicability order of its rules we could define a Grammar as a system $G = (V_N, V_T, P, N_p)$ where $N_p = \{\text{first rule tested can be any rule with LHS} = S\}$ in the case of a generative Grammar; or $N_p = \{\text{if the Algorithm terminates, the last rule applied must be a rule with RHS} = S\}$, in the case of a Recognizing Grammar.

Thus in the case of a Generative Grammar the start symbol S is transformed (by successive applications of the productions) to a string belonging to the Language which can be generated by the Grammar, while in the case of a Recognizing Grammar, a string which belongs to the Language recognized by the Grammar is transformed to the ending symbol S .

The symbol " $\xrightarrow{*}$ " connects two strings if the one written to its left can be transformed to the other, written to its right, by a number of applications of a number of productions. In the case in which the first string can be transformed to the second by a single application of a production, the symbol " \rightarrow " is used instead.

It has been proved¹ that for every Generative, respectively, Recognizing Grammar there exists its "Dual" Recognizing, respectively, Generative Grammar. These two Grammars are equivalent in the sense that the Recognizing one accepts only strings which can be generated by its Dual Generative; and that the Generative one generates only strings which can be accepted by its Dual Recognizing. By $L(G)$ we symbolize the language generated respectively recognized by a Generative, respectively, Recognizing Grammar.

Thus if $L(G_g)$ is the Language generated by the Grammar $G_g = (V_N, V_T, P, S)$ and $L(G_r)$ is the Language recognized by its dual G_r , we will have:

$$G_r = (V_N, V_T, P^{-1}, S) \quad \text{where:}$$

$$\forall (z \rightarrow y) \in P \quad \text{we have:} \quad (y \rightarrow z) \in P^{-1}, \text{ with } z \in V^+, y \in V^*.$$

Thus:

1 Arto Salomaa: "Formal Languages".

$$L(G_g) = \{\Omega/\Omega \in V_T^*, S \not\rightarrow^* \Omega\}$$

$$L(G_r) = \{\Omega/\Omega \in V_T^*, \Omega \not\rightarrow^* S\}$$

and
$$L(G_g) = L(G_r) .$$

5.1 Types of Grammars

By establishing restrictions on the productions of the previously defined Grammar G we have the following types of Grammars.

a. A type 0 Grammar is the same as Grammar G . That is an unrestricted Grammar.

b. A type 1 or Context-sensitive Grammar is a type 0 Grammar whose every production " $a \rightarrow b$ " is restricted so that:

$$|a| \leq |b| \quad \text{if the Grammar is Generative;}$$

$$|a| \geq |b| \quad \text{if the Grammar is Recognizing.}$$

c. A type 2 or Context-free Grammar is a restricted type 1 Grammar such that for every one of its productions " $a \rightarrow b$ " we have

$$a \in V_N \quad \text{and} \quad b \in V^+ \quad \text{if the Grammar is Generative;}$$

$$b \in V_N \quad \text{and} \quad a \in V^+ \quad \text{if the Grammar is Recognizing.}$$

d. A type 3 or Regular Grammar is a restricted type 2 Grammar such that every one of its productions will be:

$$a \rightarrow aB \quad \text{or} \quad C \rightarrow b \quad \text{for a Generating Grammar;}$$

$$aB \rightarrow A \quad \text{or} \quad b \rightarrow C \quad \text{for a Recognizing Grammar;}$$

$$\text{where } A, B, C \in V_N \quad \text{and} \quad a, b \in V_T.$$

6. GRAMMATICAL DEVICES

The word acceptor is usually connected with an automaton and the word generator with a grammar, although it is known that automata,

respectively, grammars can be generative, respectively, recognizing devices.

Since for every device generating a language, there is a recognizing¹ one for the same language, and vice versa, we will use the term "Grammatical Devices" for every device which can generate or recognize a Language.

Thus we define as a Grammatical Device a system GM, such that:

$$GM = (\Sigma, \Pi, T) \quad \text{where:}$$

- Σ : is a finite set of input or terminal characters (or symbols).
- Π : is a finite set of all used characters and $\Sigma \subseteq \Pi$.
- T : is the whole function, of the device, which may be simple or complicated depending on the different classes of devices. Every action of the Grammatical Device following any input of information and/or an internal instruction (included in T) is included in T. T also includes the criteria for the acceptance of an input string and/or the termination criteria after the generation of a string. The function T is defined for any Grammatical Device.

Examples of Grammatical Devices:

A Turing Machine TM is defined as the Grammatical Device:

$$TM = (\Sigma, \Gamma, T) \quad \text{where} \quad \Gamma \equiv \Pi \quad \text{and}$$

$$T = (\bar{K}, \delta, q_0, F) .$$

A Markov Algorithm MA is defined as the Grammatical Device:

$$MA = (\Sigma, A, T) \quad \text{where} \quad A = \Pi \quad \text{and}$$

$$T = (P, N_p) .$$

1. It is in fact a "semirecognizing".

CHAPTER I

FOR EVERY MARKOV ALGORITHM THERE IS AN
EQUIVALENT TURING MACHINE

We shall prove the proposition enunciated in the title of this chapter by constructing a Turing Machine, "TM", for an arbitrary Markov Algorithm "MA", which will be able to do exactly what MA can do and no more.

1. INTRODUCTION

As it is well known, when MA is given a string w , as input, it works in the following way:

- a. 1. The search for the applicability of the rules starts with rule 0.
2. After the application of a non terminating rule, the next rule tested is rule 0. After the application of a terminating rule, the Algorithm terminates.
3. If the rule K was tested and found inapplicable, the next rule tested is the rule $K + 1$. If however rule K is the last of the MA rules, MA terminates.

In the case of TM, the same kind of search must be followed, although the process will be more complicated. Thus after the application of an applicable rule by the corresponding moves of TM, or after the inapplicability of a rule has been decided by a TM process corresponding to that of MA, the head of TM will point to the first character of the string while TM will be in a suitable state for the applicability search of the next rule being tested.

It can be easily seen that for an MA rule, there will be more than one corresponding TM moves. Therefore it is appropriate to symbolize the states of TM by $Q(i, j)$ where i denotes the i MA rule and j will be an integer ($0 \leq j \leq u$)¹ where u is the greatest second subscript for the states which are used in the TM moves corresponding to rule i . If the number of MA rules is n , then for i we must have: $0 \leq i \leq n-1$. But we allow for i to extend up to: $i \leq n+2$.

¹ See section 4.4.

We do that because, as it will be explained later, the states $Q(n, j)$, $Q(n+1, j')$ and $Q(n+2, j'')$ will be used in some sets of TM moves, not corresponding to MA rules, but indispensable for the construction of TM.

Another complicated situation relating to the moves of TM corresponding to the application of an MA rule, arises when the lengths of RHS and LHS are not equal. (The length of a string t is symbolized by $|t|$). Thus if $|LHS_i| < |RHS_i|$ for the MA rule i , the tape must be "expanded" by a number of squares equal to the difference $|RHS_i| - |LHS_i|$. If $|LHS_i| > |RHS_i|$, a number of squares equal to the difference $|LHS_i| - |RHS_i|$ must be "removed" from the tape of TM.

After the above preliminary explanations we are now ready to define and construct TM.

2. DEFINITION OF TM

Let us suppose that MA is defined by:

$$MA = (A, \Sigma, P, N_p)$$

where $\Sigma \subseteq A$ is the finite set of input symbols (or characters).

A is the alphabet of MA, that is the finite set of all symbols used in MA.

P is a finite set of rules, with n elements.

N_p is a finite set of instructions for the next rule tested.

Then, assuming that the tape of TM will be infinite to the right only, TM will be defined by:

$$TM = (\bar{K}, \Gamma, \Sigma, \delta, Q(0,0), F).$$

where: \bar{K} is the set of TM states

$$\Gamma = (A \setminus \{B, X, Y, S, \#\} \cup T \cup D)$$

$$\text{where: } T = \{T_0, T_1, \dots, T_{n-1}\}$$

$$D = \{D_0, D_1, \dots, D_{m-1}\}$$

$$\text{with } m > 0, m = \max_{0 \leq i < n} (|RHS_i| - |LHS_i|)$$

δ is a mapping function from:

$$\bar{K} \times \Gamma \text{ to } \bar{K} \times (\Gamma - \{B\}) \times \{L, R\}$$

where for all the moves of the type

$$\delta\{Q(i, w_i), \Omega_i\} = \{Q(\theta_i, 0), T_{\theta_i}, R\}, \quad \theta_i$$

is the next rule tested according to the instruction set N_p .

Here $Q(i, w_i), Q(\theta_i, 0) \in \bar{K}$ and $\Omega_i, T_{\theta_i} \in \Gamma$.

$Q(0, 0)$ is the initial state of TM.

F is the set of final states. Actually only one final state is needed

$$F = \{Q(n+2, 0)\}.$$

3. AN OUTLINE OF THE CONSTRUCTION

In the following construction of TM, it will be assumed that an algorithmic machine, the MA algorithmic machine, is simultaneously switched on for a given input, and that it remains available during the entire process of construction. It will be also supposed that after every action of MA and before its next action, MA will temporarily pause. During this MA pause, in addition to already (if it is not the very beginning) existing moves and states of TM,

a number (possibly zero) of moves and a number (possibly zero) of states, both necessary for the TM action corresponding to that of MA, will be constructed and stored in TM.

Thus we see that everything necessary for TM in order to enable it to act as MA did in its last action, is given to TM. So TM performs its next move (or moves) simulating MA, i.e. producing the same result as MA.

(The kind of MA actions we mention here will be the smallest ones considered independent and not further divisible. E.g.: "A character of the string is compared with one of the characters of the LHS of an MA rule". This is an independent action and cannot be divided further.)

This method of TM construction will continue until every kind of string, even the most "peculiar" ones have been given as input to MA and TM. The main idea is that TM will simulate MA in its previous action and simultaneously will be provided with everything necessary for its next action.

In other words TM will be provided at each step with only the amount of knowledge necessary for its next action. This amount of knowledge which is "empty" at the beginning of the construction, will reach its maximum when TM is able to do exactly what MA can do.

If now the input string for MA is $w \in \Sigma$, the tape of TM will be:

SwBBB...

with its head scanning the first character of the tape, "S", and TM being in state $Q(0,0)$.

4. CONSTRUCTION OF TM

4.1 Introduction

For our MA we assume the following:

1. Set A is finite with K elements

$$A = \{a_1, a_2, \dots, a_K\}$$

2. MA has n rules.

3. The length of LHS of a rule i is r and is symbolized by $|LHS_i|$

The length of RHS of a rule i is s and is symbolized by $|RHS_i|$

$$\text{So: } |LHS_i| = r, \quad |RHS_i| = s$$

4. Rule i is: $a_{i_1} a_{i_2} \dots a_{i_r} \rightarrow a_{j_1} a_{j_2} \dots a_{j_s}$

where $a_{i_1}, a_{i_2}, \dots, a_{i_r}, a_{j_1}, a_{j_2}, \dots, a_{j_s} \in A$.

For TM

Special treatment of a blank "character"

According to our constrain for Turing Machines, a B (blank symbol) can only be read but never typed. Although we want to maintain this restriction in general, nonetheless, whenever the head going right (exclusively with a view to getting the information "where the string finishes") scans the first square with a B, we allow for a "B" to be printed again in this square. This is purely for convenience and to avoid having to duplicate all rules involving a single B as the rightmost character of the LHS. Of course we never allow for a "B" to be printed on a square where any other character was before.

4.2 First TM move included

At the very beginning of the program, and as in every Turing Machine, the head of TM is scanning the first square of the tape. As we said before, in the first square of TM's tape there is an "S".

At this moment TM is in state $Q(0,0)$ and for the pair $Q(0,0)$, S we include the move:

$$\delta\{Q(0,0), S\} = \{Q(0,0), T_0, R\}$$

4.3 Initial and Final TM moves for every MA rule.

As was explained before, the first, and only once applicable, move of TM is:

$$\delta\{Q(0,0), S\} = \{Q(0,0), T_0, R\}$$

After this move, the search for the applicability of rule zero starts. If then (after an exhaustive search for the existence of LHS_0 in the string) the LHS_0 will be found somewhere in the string, TM will apply the rule "0" using its corresponding set of moves for the application. If, however, the LHS_0 is not in the string the rule will not be applied (as inapplicable in this case).

In both cases the head of TM will return to the beginning of the tape looking for the symbol T_0 . But the state of TM will be different in the two cases. Thus if the rule is inapplicable TM will be in state $Q(0, w+1)$, and if the rule has been applied TM will be in state $Q(0, w)$.¹

So if rule 0 is inapplicable:

$$\delta\{Q(0, w+1), T_0\} = \{Q(1, 0), T_1, R\} \text{ if MA has more than 1 rule.}$$

¹ Subscripts $w, w+1, w+2$ will be explained in the sequel.

If the rule has been applied:

$$\delta\{Q(0,w), T_0\} = \{Q(\theta,0), T_\theta, R\}$$

where w is the greater (apart, obviously, from the subscripts, " $w+1$ ", " $w+2$ "...) second subscript for the TM states which correspond to the MA rule zero and have as first subscript the integer zero. Generally, $\forall i, \theta: 0 \leq i < n, 0 \leq \theta < n$ we will have:

$$\delta\{Q(i,w), T_i\} = \{Q(\theta,0), T_\theta, R\} \text{ if rule } i \text{ was applied}$$

if $|LHS_i| > |RHS_i|$ as we shall see it will be:

$$\delta\{Q(i,w), Y\} = \{Q(\theta,0), T_\theta, R\}. \text{ And } \forall i: 0 \leq i < n-1$$

$$\delta\{Q(i,w+1), T_i\} = \{Q(i+1,0), T_{i+1}, R\} \text{ if rule } i \text{ was found inapplicable.}$$

If, in the latter case, $i = n-1$, which means that the last, that is $(n-1)^{th}$, MA rule is inapplicable, MA would terminate and so TM must halt. Thus for this case we include the following move:

$$\delta\{Q(i,w+1), T_i\} = \{Q(n+2, 0), S, R\}$$

which will be a final move for TM. The state $Q(n+2, 0)$ is the final state of TM and as we shall see it can be also reached from the state $Q(i,w)$.

4.4 Meaning of subscript " w "

If the MA rule i is found to be applicable and applied, by the corresponding moves of TM, TM will be in a state $Q(i,w)$ after the substitution of LHS_i by RHS_i has been completed. So TM will be in state $Q(i,w)$ when the head is scanning the character T_i .

If the rule i is found to be inapplicable, TM will be in state $Q(i,w+1)$ when the symbol T_i is scanned. Evidently $Q(i,w+1)$ is not a next state for $Q(i,w)$; the first means that rule i was inapplicable and the second that it was applied. The only reason we use a 1θ is defined in section 4.5.

greater subscript $w+1$ (and $w+2$ as we shall see in the sequel) than the subscript w , is that in case the rule i is applicable the succession of the TM moves and states will be more readable. (Actually the second subscripts are successive integers if the rule is applicable, e.g. $Q(i,1), Q(i,2) \dots Q(i,w)$).

The subscript w will be an expression of the lengths of LHS_i and RHS_i as we shall see, $\forall i: 0 \leq i < n$. This of course means that the number of states for every MA rule i will be a function of $|LHS_i|$ and $|RHS_i|$. As it has been explained w is uniquely determined for every rule i and the w 's of two or more rules may be different. Thus, whenever we talk about rules i, j, k, \dots we will use corresponding subscripts for their w 's: $w_i, w_j, w_k \dots$.

4.5 Meaning of θ

If rule i is not a terminating rule, then the first of the above moves is the last for this rule and for θ we will have:

$$0 \leq \theta < n.$$

θ (an integer) is the label of the next rule tested, if MA is labelled.¹ If MA is unlabelled $\theta = 0$.

In the case in which rule i is a terminating rule: $\theta = n+2$ and instead of T_θ an "S" will be used, as in the beginning. Also there will not be a next state for $Q(n+2,0)$ and no move defined for state $Q(n+2,0)$, because this is the final state of TM. In this case this very last move of TM will be:

$$\delta\{Q(i,w), T_i\} = \{Q(n+2,0), S, R\}$$

$$(\text{if } |LHS_i| > |RHS_i| \rightarrow \delta\{Q(i,w), Y\} = \{Q(n+2,0), S, R\})$$

After this move TM will be in state $Q(n+2,0)$, the head will point to the first character of the string and the tape will look like:

1 A labelled MA is defined on page 102.

Head

\neq	...	\neq	\neq	S	a_{σ_0}	a_{σ_1}	...	a_{σ_w}	B	B	B	...
--------	-----	--------	--------	---	----------------	----------------	-----	----------------	---	---	---	-----

finite number
of "unwanted"
and unused
squares

modified string

blank squares

Up to now we have included the very first move and the last moves of TM for every MA rule applicable or not.

When the search for the applicability of the i th MA rule starts, TM is in state $Q(i,0)$ and its head is scanning the square with the first character of the string. In what it follows, it will be described how the search for the applicability of rule i is conducted and, in case that the rule i is applicable how it is applied.

4.6 Applicability search for rule i ($\forall i: 0 \leq i < n$)

When the search for the applicability of a rule i starts, TM is in state $Q(i,0)$ and its head is scanning the square with the first character of the string.

If now for some reason (e.g. LHS_i does not appear in the string or the string at this stage is empty) the head of TM is scanning a blank square while TM is in state $Q(i,0)$, this means that rule i is inapplicable and the head must return to the beginning looking for a square with the T_i . Going to the left, the head will ignore everything else, reading and printing the symbol it finds in every square, where there is not a T_i . At this time TM will be in the state $Q(i,w+1)$. w is an integer and its value will be given later.

Thus, the following moves are included:

$$\delta\{Q(i,0), B\} = \{Q(i,w+1), B, L\}$$

$$\delta\{Q(i,w+1), \xi\} = \{Q(i,w+1), \xi, L\} \quad \forall \xi \in A \quad (A = a_1, \dots, a_K) .$$

The move: $\delta\{Q(i, w+1), T_i\} = \{Q(i+1, 0), T_{i+1}, R\}$ as we have already mentioned will be included only if $i < n-1$. If $i = n-1$, it means that the last MA rule was found inapplicable and so MA must halt. This is, obviously, possible only for the last MA rule and the following move is included only in the TM moves corresponding to the last MA rule:

$$\delta\{Q(i, w+1), T_i\} = \{Q(n+2, 0), S, R\}$$

After this move the head will point to the first character of the modified string and the tape will have the form:

$$\neq * Sw' BBB \dots$$

There is a finite number of squares with \neq 's to the left of S (possibly zero), the meaning of which will be explained later.

Apart from the case where the machine is in state $Q(i, 0)$ and scanning a "B" there are two other eventualities open to TM as found in the same state.

a_0 : the scanned character is not the same with the first in LHS_i

$$\delta\{Q(i, 0), \xi_{i_1}\} = \{Q(i, 0), \xi_{i_1}, R\} \quad \forall \xi_{i_1} \in (A - \{a_{i_1}\})$$

b_0 : the scanned character is the same with the first in LHS_i

$$\delta\{Q(i, 0), a_{i_1}\} = \{Q(i, 1), S_{a_{i_1}}, R\} \quad \text{where } S_{a_{i_1}} \in \Gamma, S_{a_{i_1}} \notin A,$$

is a special symbol (marker).

This move is applied only if $r > 1$.

In case $r=1$ we do not use the marker $S_{a_{i_1}}$.

So in case (a_0) a number of $K-1$ moves is included; one move for every character in A but a_{i_1} . In case (b_0) there is only one move included.

So we have included all the moves of TM, when TM is in the state $Q(i,0)$.

If now the first character of LHS_i has been found in the tape of TM and the very next character in the tape is scanned there are two cases again.

a_1 : The scanned character is not the same as the second in LHS_i

(K-1 moves): $\delta\{Q(i,1), \xi_{i_2}\} = \{Q(i, w+2), \xi_{i_2}, L\} \quad \forall \xi_{i_2} \in [(A - \{a_{i_2}\}) \cup \{B\}]$

In this case the head goes left looking for the marker $S_{a_{i_1}}$ which is in the first square to the left. As it is evident from the above move, if the word in LHS_i exists at all in the tape, it cannot start from the square where the marker $S_{a_{i_1}}$ is. So the character which existed previously in this square has to be typed again by the move:

(one move) : $\delta\{Q(i, w+2), S_{a_{i_1}}\} = \{Q(i, 0), a_{i_1}, R\}$

TM is now in state $Q(i,0)$ again and looking for another a_{i_1} .

b_1 : The scanned character is the same with the second in LHS

(one move) : $\delta\{Q(i,1), a_{i_2}\} = \{Q(i,2), a_{i_2}, R\}$

After the above move TM found in state $Q(i,2)$, is looking for the next character in LHS.

Apart now from cases a_0 and b_0 which are possible when TM is in state " $Q(i,0)$ " and looking for the first character of LHS_i , and cases a_1, b_1 , we further include as our next step the following sets of moves.

If TM is in state $Q(i, i')$ where:

$$1 < i' \leq r-2$$

for each i' , we include one move of the type:

$$\delta\{Q(i, i'), B\} = \{Q(i, w+2), B, L\};^1$$

we also include (for each i') $K-1$ moves of the type:

$$\delta\{Q(i, i'), \xi_{i' + 1}\} = \{Q(i, w+2), \xi_{i' + 1}, L\} \quad \forall \xi_{i' + 1} \in (A - \{a_{i' + 1}\});$$

and the move (for each i')

$$\delta\{Q(i, i'), a_{i' + 1}\} = \{Q(i, i' + 1), a_{i' + 1}, R\}$$

4.7 Last character of LHS (here $r > 1$)

Let us suppose now that all but the last character of LHS_i has been found somewhere in the tape in immediate succession. So TM must be in state $Q(i, r-1)^2$ and looking for the character " a_{i_r} ". If this character will be found in the very next move of TM (if it is

1 After the move the machine is in state $Q(i, w+2)$ and not $Q(i, w+1)$. This happens because the marker $S_{a_{i_1}}$ must be substituted again by a_{i_1} . The disadvantage here is that although it is known that the length of LHS is greater than the remaining unsearched part of the tape, TM, in state $Q(i, 0)$, will nevertheless start to search again that part.

This of course can be avoided if we use another state $Q(i, w+3)$ $\forall i: 0 \leq i < n$, and the type of moves:

$$\delta\{Q(i, i'), B\} = \{Q(i, w+3), B, L\}$$

$$\delta\{Q(i, w+3), \xi\} = \{Q(i, w+3), \xi, L\} \quad \forall \xi \in A$$

$$\delta\{Q(i, w+3), S_{a_{i_1}}\} = \{Q(i, w+1), a_{i_1}, L\}$$

We do not intend to use this method; in our own way, with one state less, especially if alphabet A is very big, a lot of moves will be avoided. However, in case alphabet A is not too big and the length of LHS_i is "too" big the use of state $Q(i, w+3)$ must be considered.

2 The last set of General Moves in section 4.6 was:

$$\delta\{Q(i, i'), a_{i' + 1}\} = \{Q(i, i' + 1), a_{i' + 1}, R\}, \quad \forall i: 1 \leq i' \leq r-2.$$

in the next square to the right) this means that the LHS_i will have been found and the procedure for substituting it by the RHS_i will start automatically. If, however, the character in the next square is not the "wanted" one, the head of TM will have, going left, to restore character a_{i_1} in its former place presently occupied by the marker $S_{a_{i_1}}$, and to look for another a_{i_1} to the right of that position. So the process will be repeated until either LHS_i is found somewhere in the tape or until a blank symbol is met.

Consequently, we further include the following set of moves:

I. Failure

The next to the right square is a B:

$$\delta\{Q(i, r-1), B\} = \{Q(i, w+2), B, L\} \quad \underline{\text{one move}}$$

The next to the right square is not B or a_{i_r}

$$\delta\{Q(i, r-1), \xi_{i_r}\} = \{Q(i, w+2), \xi_{i_r}, L\} \quad \underline{K-1 \text{ moves}}$$

$$\forall \xi_{i_r} \in (A - \{a_{i_r}\})$$

II. Success

From the following set of moves, obviously, only one will be constructed for each MA rule:

$$\underline{\text{Case A}} \quad |LHS_i| = |RHS_i| \quad (r=s)$$

For this case when the last character of LHS_i has been found, it is immediately replaced by the last character of RHS_i and the head goes left for the other substitutions. We can see that the move is applicable even for the case where $r=1$.

$$\delta\{Q(i, r-1), a_{i_r}\} = \{Q(i, r), a_{j_s}, L\}$$

$$\underline{\text{Case B}} \quad |LHS_i| < |RHS_i| \quad r < s \quad (r > 0)$$

We will make a distinction here for $r=1$ and $r>1$, since there must be different moves for every case.

$$r=1 : \delta\{Q(i,0), a_{i_1}\} = \{Q(\sigma,0), D_p, R\};$$

state $Q(\sigma,0)$ and symbol $D_p \in D$ will be explained in "case $r<s$ ".

$$r>1 : \delta\{Q(i,r-1), a_{i_r}\} = \{Q(i,r), X, L\}$$

Case C $|LHS_i| > |RHS_i| \quad (r>s)$

$$r>1 : \delta\{Q(i,r-1), a_{i_r}\} = \{Q(i,r), b, L\}$$

$$\text{where } b = \begin{cases} a_{j_s} \in A & \text{if } s>0 \\ Y \notin A & \text{if } s=0 \end{cases}$$

$$r=1 \text{ (i.e. } s=0)$$

$$\delta\{Q(i,0), a_{i_1}\} = \{Q(\tau,0), Y, L\}.$$

State $Q(\tau,0)$ and symbol Y will be explained in "case $r>s$ ".

4.8 Substitution

At this stage the search for the applicability of rule i has finished, as the LHS_i was found in the tape of TM. Therefore, the substitution which has already started, for the case $r=s$, will take place in the tape of TM. Since there are in general three distinct possibilities for MA rules ($r=s$, $r<s$, $r>s$), we shall include a different set of TM moves for every case. Of course for every MA rule only one case will be possible and so only the corresponding set of moves will be included.

4.8.1 Case A: substitution for $r=s$

The head of TM now will go left printing the characters of RHS_i from right to left. Since at this stage it is known which character

will be read by TM (the characters of LHS_i), only one "next move" will be defined and included, as long as the substitution will be taking place.

Thus the following set of moves is included provided that $r > 1$,

$$\delta\{Q(i, r+i'), a_{i_{r-(i'+1)}}\} = \{Q(i, r+i'+1), a_{j_{s-(i'+1)}}, L\}$$

$$\forall i' : 0 \leq i' \leq r-3$$

(It is to be noted that we use a different state, ("next state") after every substitution of a character, because some characters may appear in LHS_i more than once).

The last move of the substitution is the one which makes TM print a_{j_1} (the first character of RHS_i) in the square where the marker $S_{a_{i_1}}$ is. This move is next to the last of the above moves. (For $i'=r-3$, next state is $Q(i, 2r-2)$).

So for $r > 1$ again:

$$\delta\{Q(i, 2r-2), S_{a_{i_1}}\} = \{Q(i, 2r-1), a_{j_1}, L\}$$

After this move or after the move $\delta\{Q(i, r-1), a_{i_r}\} = \{Q(i, r), a_{j_s}, L\}$ or $\delta\{Q(i, 0), a_{i_0}\} = \{Q(i, 1), a_{j_1}, L\}$ for $r=1$ which has been already included, the i th MA rule has almost been completely initiated. We say "almost", because the head of TM must return to the beginning of the string, or rather to the square with a " T_i " which is to the left of the first character of the string. Every character in the tape will obviously remain the same at this stage.

Thus, we include a move for every character in A and for the same state $Q(i, 2r-1)$ of TM.

$$\delta\{Q(i, 2r-1), \xi\} = \{Q(i, 2r-1), \xi, L\} \quad \forall \xi \in A^1$$

1 Here $r=s$ and $2r-1 = r+(s-1)$; so state $Q(i, 2r-1)$ can be written as $Q(i, r+(s-1))$

Finally we include the move:

$$\delta\{Q(i, 2r-1), T_i\} = \{Q(\theta, 0), T_\theta, R\}$$

where θ is an integer $0 \leq \theta < n$, such that if MA is labelled, θ is the label of the next rule tested. If, however, MA is unlabelled, $\theta=0$. In case rule i is a terminating rule:

$$\theta=n+2 \quad \text{and} \quad T_\theta \text{ will become } S$$

This will be adequately explained later in detail.

4.8.2 Case B substitution for $r < s$

$r > 1$: In this case TM will substitute first by moving from right to left, all the characters of LHS_i (except for the first, this being the marker S_{ai_1}) by X's. A symbol D_p will be printed in the place of S_{ai_1} .

$r=1$: In this case the unique symbol of LHS (not a marker) will be substituted by the symbol D_p , and TM will enter in a state $Q(\sigma, 0)$, by the move

$$\delta\{Q(i, 0), a_{i_1}\} = \{Q(\sigma, 0), D_p, R\}$$

which has been already included in our construction.

The substitution of LHS_i by X's will be done in the following way:

TM, being in state $Q(i, r)$, will not change state before the marker S_{ai_1} is read. Thus TM will remain in state $Q(i, r)$ as long as its head is going left, printing X's, in the squares where the characters of LHS_i are to be found. We can easily see that the number of the moves which are needed for this substitution will be equal or less to the number of characters in LHS_i . (Actually, we shall include one move for each distinct character of LHS. That means that if there are repetitions of some characters in LHS_i , only one move will be included for these "same" characters.) This is so because for each

move $\delta\{Q(i,r), \xi_{LHS_i}\} = \{Q(i,r), X, L\}$, where ξ_{LHS_i} is a character in LHS_i (and $\xi_{LHS_i} \neq s_{a_{i1}}$), the next state will be the same, the character printed the same and the direction of the head the same.

Thus the following set of moves should be included:

$$\delta\{Q(i,r), \xi_{LHS_i}\} = \{Q(i,r), X, L\} \quad \forall \xi_{LHS_i} \text{ a character of } LHS_i, \text{ with } \xi_{LHS_i} \neq a_{i1} \text{ (if there is not a repetition of } a_{i1} \text{ in } LHS_i)$$

We also have to include one move of the form:

$$\delta\{Q(i,r), s_{a_{i1}}\} = \{Q(\sigma, 0), D_p, R\}$$

where p is an integer, $p = s - r - 1$

4.8.2.1 Explanation of D_p

MA has n rules. We examine all the positive differences $s_i - r_i$, where s_i and r_i represent the length of LHS_i and RHS_i of the i th MA rule respectively for all $i: 0 \leq i < n$. For some i , there will be a maximum difference $(s_i - r_i)$, or a difference $(s_i - r_i)$ which is not smaller than any other $\forall i: 0 \leq i < n$. Suppose then that for this i : $s_i - r_i = m$. Obviously $m > 0$.

We construct now a set D , which will be ordered and will have m elements.

$$D = \{D_0, D_1, \dots, D_{m-1}\}$$

Each one of the above particular symbols D_0, D_1, \dots, D_{m-1} will have a special meaning which is to be explained presently.

When TM is "working" on the i th MA rule, for which $|LHS_i| < |RHS_i|$, if the LHS_i string is found somewhere in the tape, a number of squares equal to the difference $(|RHS_i| - |LHS_i|)$ must be added to the squares of LHS_i in order that sufficient space for the "nesting" of RHS_i be

created.

One way of achieving this "expansion" of the tape of TM, is by moving every character lying between the last character of LHS_i and the first blank symbol, so many places to the right as the difference $(|RHS_i| - |LHS_i|)$ determines. The symbol X is to be printed on the newly available squares as well as on the squares where the LHS_i is located.

This operation will be performed through the use of a special routine by TM. This same routine will be employed for every MA rule with $|RHS| > |LHS|$. By utilising this routine we avoid using a greater number of states for every single MA rule of the said type.

4.8.3 Routine "Expansion"

When TM starts the execution of the routine, it is in state $Q(\sigma, 0)$, where $\sigma = n$. At the first move TM prints the symbol $D_p \in D$.

The symbol D_p , where $0 \leq p \leq m-1$, corresponds to the difference $s_i - r_i$ (where i is the MA rule, TM is working on). Because D_{m-1} corresponds to the maximum of all positive differences $s_i - r_i$, $\forall i : 0 \leq i < n$, there will be a corresponding symbol D_p in D , for every positive difference $s_i - r_i$, $0 \leq i < n$. Thus we can say that there exists a mapping from the set of positive differences $s_i - r_i$ into the set $D = \{D_0, D_1, \dots, D_{m-1}\}$.

If now for the rule i we have $s_i - r_i = p > 0$, the corresponding element (or the image) in D will be D_{p-1} ; and the first move when TM starts the execution of the routine will be:

$$\delta\{Q(i, r), s_{a_{i1}}\} = \{Q(\sigma, 0), D_{p-1}, R\} \quad \text{if } r > 1$$

$$\delta\{Q(i, 0), a_{i1}\} = \{Q(\sigma, 0), D_{p-1}, R\} \quad \text{if } r = 1 \quad (\text{this move}$$

has been already included).

All the characters now, between D_{p-1} and the first blank symbol, are moved one square to the right in such a way that the last character of the string is printed on the blank square immediately to its right, the penultimate on the square where the last was and so on. Finally an X is printed on the square to the right of character D_{p-1} .

Following that, D_{p-1} is replaced by D_{p-2} ($p \geq 2$), and this process is repeated until the symbol D_0 is read by TM. Then D_0 will be replaced by X and at this stage the number of successive X's will be equal to the length of RHS_i .

The next step required is the substitution of these X's by RHS_i . But TM, being in a state $Q(\sigma, r)$ of the routine must regain the state it was in before it "entered" into the execution of the routine. Therefore the head must go left ignoring everything except a T_i $\forall i : 0 \leq i < n$. As soon as one of these characteristic symbols (for every MA rule) has been read, TM is again found in the last state it was before changing to state $Q(\sigma, 0)$. What the head has to do now is to go right ignoring everything except the first X to meet in its way. When the first X is read, the substitution of the string of X's by the RHS_i takes place.

Example of the tape after LHS_i has been found

(Here $s_i - r_i = 3$; thus corresponding symbol is D_2)

LHS_i

$T_i \ a_{K_1} \ a_{K_2} \dots a_{\lambda} \ S_{a_{i_1}} \ a_{i_2} \dots a_{i_r} \ a_{\mu_1} \ a_{\mu_2} \dots a_r \ B \ B \ B \dots$
 $T_i \ a_{K_1} \ a_{K_2} \dots a_{\lambda} \ D_2 \ X \dots X \ a_{\mu_1} \ a_{\mu_2} \dots a_r \ B \ B \ B \dots$
 $T_i \ a_{K_1} \ a_{K_2} \dots a_{\lambda} \ D_2 \ X \dots X \ X \ a_{\mu_1} \ a_{\mu_2} \dots a_r \ B \ B \ B \dots$
 $T_i \ a_{K_1} \ a_{K_2} \dots a_{\lambda} \ D_1 \ X \dots X \ X \ a_{\mu_1} \ a_{\mu_2} \dots a_r \ B \ B \ B \dots$
 $T_i \ a_{K_1} \ a_{K_2} \dots a_{\lambda} \ D_1 \ X \dots X \ X \ X \ a_{\mu_1} \ a_{\mu_2} \dots a_r \ B \ B \ B \dots$
 $T_i \ a_{K_1} \ a_{K_2} \dots a_{\lambda} \ D_0 \ X \dots X \ X \ X \ a_{\mu_1} \ a_{\mu_2} \dots a_r \ B \ B \ B \dots$
 $T_i \ a_{K_1} \ a_{K_2} \dots a_{\lambda} \ D_0 \ X \dots X \ X \ X \ X \ a_{\mu_1} \ a_{\mu_2} \dots a_r \ B \ B \ B \dots$
 $T_i \ a_{K_1} \ a_{K_2} \dots a_{\lambda} \ X \ X \dots X \ X \ X \ X \ a_{\mu_1} \ a_{\mu_2} \dots a_r \ B \ B \ B \dots$

number of X's equal
 to $|LHS_i| + 3$ which is
 equal to $|RHS_i|$

4.8.3.1 Moves of the routine "Expansion" included

$\delta\{Q(\sigma, 0), X\} = \{Q(\sigma, 0), X, R\}$ one move
 $\delta\{Q(\sigma, 0), \xi\} = \{Q(\sigma, 0), \xi, R\} \quad \forall \xi \in A, \quad A = \{a_1, a_2, \dots, a_K\}$ so
K moves
 $\delta\{Q(\sigma, 0), B\} = \{Q(\sigma, 2), X, L\}$ one move
 $\delta\{Q(\sigma, 1), X\} = \{Q(\sigma, 2), X, L\}$ one move

We include the following moves as well:

- 1 When the execution of the routine starts

$$\delta\{Q(\sigma, 2), a_1\} = \{Q(\sigma, 4), X, R\}$$

$$\delta\{Q(\sigma, 2), a_2\} = \{Q(\sigma, 5), X, R\}$$

.

.

$$\delta\{Q(\sigma, 2), a_K\} = \{Q(\sigma, K+3), X, R\}$$

or generally:

$$\delta\{Q(\sigma, 2), a_\lambda\} = \{Q(\sigma, \lambda+3), X, R\}$$

K moves

where $a_\lambda \in A, \lambda=1, 2, \dots, K$

$$\delta\{Q(\sigma, 2), X\} = \{Q(\sigma, 3), X, L\}$$

one move

$$\delta\{Q(\sigma, 4), X\} = \{Q(\sigma, 1), a_1, L\}$$

$$\delta\{Q(\sigma, 5), X\} = \{Q(\sigma, 1), a_2, L\}$$

.

.

.

$$\delta\{Q(\sigma, K+3), X\} = \{Q(\sigma, 1), a_K, L\}$$

or generally:

$$\delta\{Q(\sigma, \lambda+3), X\} = \{Q(\sigma, 1), a_\lambda, L\}$$

K moves

where $a_\lambda \in A, \lambda=1, 2, \dots, K$

$$\delta\{Q(\sigma, 3), X\} = \{Q(\sigma, 3), X, L\}$$

one move

Here after three successive X's have been read, the head goes left looking for a $D_p \in D$, while TM is in state $Q(\sigma, 3)$.

For every symbol in $D = \{D_0, D_1, \dots, D_{m-1}\}$ and state $Q(\sigma, 3)$, we include a separate move.

$$\delta\{Q(\sigma, 3), D_0\} = \{Q(\sigma, K+4), X, L\}$$

$$\delta\{Q(\sigma, 3), D_1\} = \{Q(\sigma, 0), D_0, R\}$$

$$\delta\{Q(\sigma, 3), D_2\} = \{Q(\sigma, 0), D_1, R\}$$

.

.

.

$$\delta\{Q(\sigma,3), D_{m-1}\} = \{Q(\sigma,0), D_{m-2}, R\}$$

But from all D_p , $0 \leq p \leq m-1$, only D_0 can be read when TM is in state $Q(\sigma,2)$. This can only happen if there was not an X to the right of D_p , when TM started the execution of the routine.

Example:

If the i th MA rule was $a \rightarrow bc$ and the tape was:

" T_i b c d c d cc BBB ..." then

after a number of moves the tape would be modified to:

... \rightarrow " T_i b c d D_0 d cc X BBB ..." \rightarrow ...

" T_i b c d D_0 X cc BBB ..."

$Q(\sigma,2)$

or

if the tape was:

" T_i ... a BBB ..." , then \rightarrow ...

... \rightarrow " T_i ... D_0 BBB ..." " T_i ... D_0 X BBB ..."

$Q(\sigma,2)$

Thus finally the following moves are to be included:

$$\delta\{Q(\sigma,3), D_0\} = \{Q(\sigma,K+4), X, L\} \quad \text{one move}$$

$$\delta\{Q(\sigma,2), D_0\} = \{Q(\sigma,K+4), X, L\} \quad \text{one move}$$

$$\delta\{Q(\sigma,3), D_p\} = \{Q(\sigma,0), D_{p-1}, R\} \quad m \text{ moves}$$

where: $D_p \in D, \forall_p : 0 \leq p \leq m-1$

After the above moves, the routine has almost finished. The head will go left (ignoring everything but a $T_i, \forall i : 0 \leq i < n$) while TM will be in state $Q(\sigma,K+4)$.

$$\delta\{Q(\sigma,K+4), \xi\} = \{Q(\sigma,K+4), \xi, L\} \quad K \text{ moves}$$

$$\forall \xi \in A \ (A = \{a_1, a_2, \dots, a_K\})$$

For a number n' , $0 \leq n' \leq n$, of MA rules the lengths of their LHS's will be smaller than the lengths of their RHS's.

Thus for the case of these MA rules alone, we include the following TM moves (one for each rule).

$$\delta\{Q(\sigma, K+4), T_i\} = \{Q(i, r'), T_i, R\} \quad (n' \text{ moves})$$

$$\forall i : 0 \leq i < n \text{ and } |LHS_i| - |RHS_i| < 0$$

(The case $|LHS_i| = 0$, is examined separately).

where r' is defined:

$$r' = \begin{cases} 1 & \text{if } r=1 \\ r+1 & \text{if } r>1 \end{cases}$$

In the case $|LHS_i| = 0$ (and obviously for $|RHS_i| > |LHS_i|$, $|RHS_i| > 0$) r' will be:

$$r' = s + K + 2 \quad (\text{see case } |LHS_i| = 0).$$

Thus, with one of the above last moves of the subroutine, TM is again in the same state in which it was before it started the subroutine execution.

An interesting feature of the routine expansion described above is that only $K+5$ states were used, where K is the number of elements in A .

4.8.4 Substitution following the routine expansion

After " T_i " has been read by TM, the head goes right ignoring everything except an X . TM is now in state $Q(i, r')$

$$\delta\{Q(i, r'), \xi\} = \{Q(i, r'), \xi, R\} \quad (K \text{ moves})$$

$$\forall \xi \in A$$

At the same time that the first X is scanned, the substitution of the string of X 's by RHS_i takes place:

$$(RHS_i = a_{j_1} a_{j_2} \dots a_{j_s})$$

$$\delta\{Q(i, r'), X\} = \{Q(i, r'+1), a_{j_1}, R\}$$

$$\delta\{Q(i, r'+1), X\} = \{Q(i, r'+2), a_{j_2}, R\}$$

.

.

.

$$\delta\{Q(i, r'+s-2), X\} = \{Q(i, r'+s-1), a_{j_{s-1}}, R\}$$

$$\delta\{Q(i, r'+s-1), X\} = \{Q(i, r'+s), a_{j_s}, L\}$$

$$\text{where } a_{j_{j'}} \in A \quad 1 \leq j' \leq s$$

Thus (more generally) the following moves are included:

$$\delta\{Q(i, r'+j), X\} = \{Q(i, r'+j'+1), a_{j_{j'+1}}, R\} \quad (s-1 \text{ moves})$$

$$\delta\{Q(i, r'+s-1), X\} = \{Q(i, r'+s), a_{j_s}, L\} \quad (\text{one move})$$

$$\text{where } 0 \leq j' < s-2$$

TM now has "applied" the i th MA rule and the head will go left ignoring everything but a T_i .

$$\delta\{Q(i, r'+s), \xi\} = \{Q(i, r'+s), \xi, L\} \quad \forall \xi \in A, \quad (K \text{ moves})$$

The final move of TM for this MA rule will be:

$$\delta\{Q(i, r'+s), T_i\} = \{Q(\theta, 0), R_\theta, R\}$$

where θ has the same meaning as in case A.

4.8.5 Case C $r > s$ ($|LHS_i| > |RHS_i|$)

In this case, because $|LHS_i| > |RHS_i|$, it is obvious that after the substitution of s characters of LHS_i by the RHS_i (if $s > 0$), a number (equal to $r-s$), of unwanted squares will remain on the tape. In

particular if $s=0$, all the squares occupied by LHS_i are unwanted. The substitution can easily be arranged so that these unwanted squares are successive. Of course if a part of the TM tape could be cut off, this would not present a problem. But this is not possible, and therefore we should follow some more realistic ideas.

4.8.5.1 The unwanted squares "are transferred" to the left of the first blank symbol "B"

In fact the squares of the tape cannot be moved, but there is an apparent move of squares when the characters being borne on them do change squares.

Thus, if all the characters located in squares between the "unwanted ones" and the first blank square with the symbol "B", are to be moved leftwards and occupy the "unwanted squares", starting from the first unwanted square to the left, the same number of unwanted squares will appear to the right of the first blank square. Since TM cannot type a blank character "B" on each of these squares, another character "E" can be used and treated exactly as "B". Of course a number of extra TM moves must be created for "E".

4.8.5.2 Pseudosymbols

A special character can be used which will be typed on every unwanted square. Once this character has been printed on a square of TM's tape, it will remain there for ever. This means that whenever the head of TM, scanning a square, reads this special symbol, it will print it again in the same square. Another property of this symbol is that it never makes the head change direction (after its having been read by the head).

Thus there will be no error if we consider that the squares containing such symbols are "ignored" whenever they are scanned by TM. We shall also assume that a number of squares containing this special

symbol (pseudosymbol) does not separate two other squares, with ordinary characters, if the squares with the pseudosymbol are the only ones between these two squares with the ordinary characters.

It must be emphasised here that although there will be defined moves of TM incorporating this pseudosymbol, we can always assume that the squares which contain it are to be ignored. This can be achieved if our TM is capable of states which are "left moving" or "right moving" only. For every state which is both "left and right moving" we construct two states the one for the "left moving" part and the other for the "right moving".

So if q_{LR} is a left and right moving state and " \emptyset " our pseudo-symbol, we will have:

$$\delta(q_{LR}, \emptyset) = (q_{LR}, \emptyset, ?)$$

We can see that this pair (q_{LR}, \emptyset) cannot define the next direction of the head as it is not known if the head is coming from left and must go right or if it is coming from right and must go left.

Instead of state q_{LR} we will then use two states, the q_L which is "left moving" and q_R which is right moving, thus:

$$\delta(q_L, \emptyset) = (q_L, \emptyset, L)$$

$$\delta(q_R, \emptyset) = (q_R, \emptyset, R)$$

For all the other defined pairs of q_{LR} with characters $\gamma \in \Gamma$ we have:

A. for $\delta(q_{LR}, \gamma) = (q_{LR}, \gamma', L)$ we include two moves:

$$1. \quad \delta(q_L, \gamma) = (q_L, \gamma', L) \quad 2. \quad \delta(q_R, \gamma) = (q_L, \gamma', L)$$

B. for $\delta(q_{LR}, \gamma) = (q_{LR}, \gamma', R)$ we include two moves:

$$1. \quad \delta(q_L, \gamma) = (q_R, \gamma', R) \quad 2. \quad \delta(q_R, \gamma) = (q_R, \gamma', R)$$

C. for $\delta(q_{LR}, \gamma) = (p, \gamma', \{\frac{L}{R}\})$, ($q_{LR} \neq p$), we include the moves:

$$1. \quad \delta(q_L, \gamma) = (p, \gamma', \{\frac{L}{R}\}) \quad 2. \quad \delta(q_R, \gamma) = (p, \gamma', \{\frac{L}{R}\})$$

For every case where q_{LR} is the next state of some state q we have:

D. for $\delta(q, \gamma) = (q_{LR}, \gamma', L)$ we include the move:

$$\delta(q, \gamma) = (q_L, \gamma', L)$$

E. for $\delta(q, \gamma) = (q_{LR}, \gamma', R)$ we include the move:

$$\delta(q, \gamma) = (q_R, \gamma', R)$$

4.8.5.3 The unwanted squares are transferred to the left of T_i

Whenever there is a number of unwanted squares in the tape of TM, all the characters between T_i and the first unwanted square (including T_i) will move right a number of squares equal to the number of the unwanted squares. The head of TM never goes to the left of T_i . So everything to the left of T_i is really "unwanted" and can be considered as nonexistent. Thus we assume that the beginning of the string in the tape is to the right of T_i and that the length of the string is equal to the number of squares between T_i and the first square with a "B" (blank square).

We will use this method in our TM construction; just as in the case $|RHS_i| > |LHS_i|$ a routine ("Expansion") was used, so another routine ("Cut off") will be used here for the implementation of the present method.

4.8.6 Case C $r > s$ ($|LHS_i| > |RHS_i|$), (application¹)

If $|RHS_i| > 0$, TM starts the substitution in the direction from right to left, printing the last symbol of RHS_i on the square where the last

1 The method described in Section 4.8.5.3.

symbol of LHS_i was found. Then it continues going left and printing the remaining characters of RHS_i in successive squares of the tape. After all the characters of RHS_i have been printed, the head goes left (one square) and prints a "Y". If, after that, $r-s = \mu > 1$, $\mu-1$ X's are printed on the preceding squares to the left which are occupied by the remaining symbols of LHS_i .

When an "X" has been printed on the square with the marker " S_{ai1} " (the square corresponding to the first character of LHS_i) TM will change state starting the execution of another routine: the routine "Cut off".

After the execution of the routine "cut off", there will be a number of successive unwanted squares between T_i and the first symbol of the string. The square to the left of the first character of the string will be occupied by a "Y". If $r-s = \mu > 1$, there will be $\mu-1$ successive unwanted squares occupied by "X's" between T_i and "Y". TM now will arrange that a " T_θ "¹ will be printed on the square occupied by "Y", and thus all the unwanted squares will be now to the left of T_θ . Before that, the character " \neq " is printed on all unwanted squares, including the square with the T_i (the old one). The character " \neq ", after T_θ has been printed in its new position (on the square with a "Y"), can never be met again and thus can never be read by TM because the head of TM never goes to the left of T_i ,
 $\forall i : 0 \leq i < n$.

1 T_θ is used instead of T_i because at this stage the application of rule i has been completed and another rule, rule θ , has to be tested as the next one. Of course θ can be i (again), if the next rule is rule i again. T_θ is the characteristic symbol of rule θ , where θ satisfies: $1 \leq \theta < n$.

4.8.7 Case C, Moves

$$\delta\{Q(i,r), a_{i_{r-1}}\} = \{Q(i,r+1), a_{j_{s-1}}, L\}$$

$$\delta\{Q(i,r+1), a_{i_{r-2}}\} = \{Q(i,r+2), a_{j_{s-2}}, L\}$$

$$\delta\{Q(i, r+(s-2)), a_{i_{r-(s-1)}}\} = \{Q(i, r+(s-1)), a_{j_1}, L\}$$

Here:

 $r > 1$ $s > 0$

With the above moves, (starting from right to left) a segment of LHS_i , equal in length with $|RHS_i|$, is substituted by RHS_i .

With the following moves a "Y" and "r-s-1" successive "X's" are printed on the squares with the remaining symbols of LHS_i .

$$\delta\{Q(i, r+s-1), a_{i_{r-s}}\} = \{Q(i, r+s), Y, L\}$$

$$\delta\{Q(i, r+s), a_{i_{r-s-1}}\} = \{Q(i, r+s), X, L\}$$

$$\delta\{Q(i, r+s), a_{i_2}\} = \{Q(i, r+s), X, L\}$$

if

 $r-s > 1$

$$\delta\{Q(i, r+s), S_{a_{i_1}}\} = \{Q(\tau, 0), X, L\}$$

The state $Q(\tau, 0)$, where $\tau = n+1$, is the first state of TM when it starts the execution of the subroutine "cut off". We notice here that before TM starts the execution of "cut off" is in state:

$$Q(i, r+s) \quad \text{if} \quad r > 1$$

$$Q(i, 0) \quad \text{if} \quad r = 1$$

The above moves can be more generally formulated:

For

$$LHS_i = "a_{i_1} a_{i_2} \dots a_{i_r}"$$

$$RHS_i = "a_{j_1} a_{j_2} \dots a_{j_s}" \rightarrow$$

$$\delta\{Q(i, (r-1) + i'), a_{i_{r-i'}}\} = \{Q(i, r+i'), a_{j_{s-i'}}, L\}$$

where $r > 1$, $s > 0$ and i' : $1 \leq i' \leq s-1$

$$\delta\{Q(i, (r-1)+s), a_{i_{r-s}}\} = \{Q(i, r+s), Y, L\}$$

$$\delta\{Q(i, r+s), a_{i_{r-i'}}\} = \{Q(i, r+s), X, L\}$$

where $r-s > 1$ and $i' : s+1 \leq i' \leq r-2$;

$$\delta\{Q(i, r+s), S_{a_{i_1}}\} = \{Q(\tau, 0), X, L\}$$

We see here again that when the head of TM goes leftwards printing X's on the squares with the remaining characters of LHS_i , TM does not change state (remaining in $Q(i, r+s)$) until the marker $S_{a_{i_1}}$ has been read.

4.8.8 Description and moves of "Cut off"

When TM starts the "Cut off" execution it is in state $Q(\tau, 0)$ and its head goes left. (Here the subscript " τ " is equal to $n+1$, just as " σ ", used for the states of "expansion", was equal to " n "). So the first square, to the left of marker " $S_{a_{i_1}}$ " (or a_{i_1} if $r=1$, $s=0$) can be either a character $a_\lambda \in A$, or the special character " T_i ". (If it is a " T_i ", it means that LHS_i was found at the beginning of the string).

If now the character to the left of " $S_{a_{i_1}}$ " is a " T_i ", TM terminates the execution of "Cut off", printing the special symbol " $\#$ " (used for unwanted squares) and changes state to $Q(i, r')$, where

$$r' = \begin{cases} r+s+1 & \text{if } r > 1 \\ 1 & \text{if } r = 1 \end{cases}$$

If the character to the left of " $S_{a_{i_1}}$ " (or " a_{i_1} ") is not " T_i " but a character $a_\lambda \in A$, TM changes state and its head goes right. (This state of TM will be unique (and characteristic) for every $a_\lambda \in A, \lambda = 1, 2, \dots, K$ as it will be explained later).

We thus include the moves:

$$\delta\{Q(\tau,0),T_i\} = \{Q(i,r'),\neq,R\}$$

$$\text{where } r' = \begin{cases} r+s+1 & \text{if } r>1 \\ 1 & \text{if } r=1 \end{cases}$$

$$\delta\{Q(\tau,0),a_1\} = \{Q(\tau,1),X,R\}$$

$$\vdots$$

$$\delta\{Q(\tau,0),a_K\} = \{Q(\tau,K),X,R\}$$

$$A = \{a_1, a_2, \dots, a_K\}$$

It is now seen that for every character in A, we use a different next state of TM.

$$Q(\tau,1) \quad \text{for } a_1$$

$$Q(\tau,2) \quad \text{for } a_2$$

$$\vdots$$

$$Q(\tau,K) \quad \text{for } a_K$$

These moves can be, generally, written as:

$$\delta\{Q(\tau,0),a_\lambda\} = \{Q(\tau,\lambda),X,R\}$$

$$\text{where } a_\lambda \in A, \lambda=1,2,\dots,K$$

When each of the above moves has been executed, TM, being in one of the states $Q(\tau,1), Q(\tau,2), \dots, Q(\tau,K)$, moves its head right "ignoring" all the X's (if it meets any) and it prints a symbol $a_\lambda \in A$, (corresponding to the state $Q(\tau,\lambda)$, $\lambda=1,2,\dots,K$, it was in) on the first square with a Y. Thus we include the moves:

$$\delta\{Q(\tau,1),X\} = \{Q(\tau,1),X,R\}$$

$$\vdots$$

$$\delta\{Q(\tau,K),X\} = \{Q(\tau,K),X,R\}$$

where

$$\delta\{Q(\tau,1),Y\} = \{Q(\tau,K+1),a_1,L\}$$

$$\{a_1, a_2, \dots, a_K\} = A$$

$$\vdots$$

$$\delta\{Q(\tau,K),Y\} = \{Q(\tau,K+1),a_K,L\}$$

or generally:

$$\begin{aligned} \delta\{Q(\tau, \lambda), X\} &= \{Q(\tau, \lambda), X, R\} \\ \delta\{Q(\tau, \lambda), Y\} &= \{Q(\tau, K+1), a_\lambda, L\} \end{aligned} \quad \begin{array}{l} a_\lambda \in A \\ \lambda = 1, 2, \dots, K \end{array}$$

After each of the last moves (second set), TM changes state to $Q(\tau, K+1)$, where K is the number of elements in A , and it moves its head left one square. On this square, where an "X" was before, a Y is printed. Next state of TM is $Q(\tau, 0)$.

$$\delta\{Q(\tau, K+1), X\} = \{Q(\tau, 0), Y, L\}$$

The head continues now going left, "ignoring" all X's while TM is in state $Q(\tau, 0)$ again.

$$\delta\{Q(\tau, 0), X\} = \{Q(\tau, 0), X, L\}$$

After the X's (if there are any) and if there is a square with a character $a_\lambda \in A$ to the left of the first X (left-most X) the process will be repeated. In fact, between T_i and the X's, only characters of the set A can exist. If after the X's there is nothing but a T_i , the subroutines execution terminates with a move which has already been included.

4.8.8.1 An example of the tape after the "Cut off" execution

After the termination of "cut off" procedure the tape will look as follows:

...	\neq	\neq	...	\neq	X	X	...	X	Y	a_{λ_1}	a_{λ_2}	...	a_{λ_m}	B	B	B	...
finite number of unwanted squares with \neq 's				a finite num- ber of squares with X's, possibly zero				the string				blank squares					

4.8.9 First moves after the termination of "Cut off"

TM is in state $Q(i, r')$ now and its head goes right printing " \neq 's" on every square with an "X". T_θ is printed on the square (unique)

with a "Y". After that, the application of rule i terminates.

$$\delta\{Q(i,r'),X\} = \{Q(i,r'),\#,R\}$$

$$\text{where } r' = \begin{cases} r+s+1 & \text{if } r>1 \\ 1 & \text{if } r=1 \end{cases}$$

$$\delta\{Q(i,r'),Y\} = \{Q(\theta,0), T_\theta, R\}$$

where " θ " has the same meaning as in case A.

Example of the tape after termination of rule i .

...	#	#	...	#	T_θ	a_{λ_1}	...	a_{λ_m}	B	B	B	...
-----	---	---	-----	---	------------	-----------------	-----	-----------------	---	---	---	-----

4.9 Special case $|LHS_i| = 0$

$$(a) \quad |RHS_i| = 0$$

If such a rule ($\Lambda \rightarrow \Lambda$ or $\Lambda \rightarrow .\Lambda$) exists between the MA rules, it will always be applicable in the beginning of the string and thus the string must not be searched.

Hence the head of TM, after the completion of the last move corresponding to the MA rule previously applied or tested and found inapplicable, goes right one square to the position where the first character of the string is.

TM now will execute its first move for the rule i . Since rule i is always applicable, this first move will be independent of what is being scanned by the head. Thus it should be arranged so that whatever character is in this square, it will be read and printed, and TM, without changing state, will drive its head one square to the left, where the special symbol " T_i " is. Then the second and last move of the rule will be the same as in the case (b) which will follow.

So the following move is included:

$$\delta\{Q(i,0),a_\lambda\} = \{Q(i,0),a_\lambda,L\} \quad a_\lambda \in A \quad \lambda = 1,2,\dots,K$$

$$(b) \quad |RHS_i| = s > 0$$

In this case the tape must be "expanded". The routine "Expansion" is not used here. Instead a number of moves, which do almost the same thing, is included. These moves "add" squares with "X's" between " T_i " and the first character of the string. The number of squares required is equal to the length of RHS_i , which is " s ". Thus, after " s " squares have been "added" and placed between " T_i " and the first character of the string, the following moves of TM, will be the same as the ones which follow the moves of routine "Expansion".

We include the following set of moves:

$$\delta\{Q(i,0),a_\lambda\} = \{Q(i,0),a_\lambda,R\} \quad a_\lambda \in A, \lambda = 1,2,\dots,K$$

In this set of moves TM "ignores" everything and its head goes right looking for the first blank square.

TM prints an "X" on the first blank square, changes state and does the same thing (the head goes right to the next blank square, prints an "X", and TM changes state) $s-1$ times, including the first. Finally the head goes right one square again, and a "Y" is printed there. Thus on s squares to the right of the last character of the string, there are $s-1$ "X's" and a "Y" now. This "Y" is in the first square to the left of the first blank square.

At this stage the tape has been expanded by s squares ($s = |RHS_i|$) but these squares have to be "transferred" to the space between the square with " T_i " and the square with the first character of the string. This means of course that all the characters between " T_i " and the first "X" have each to be transferred s squares to the right.

$$\begin{aligned}
\delta\{Q(i,0),B\} &= \{Q(i,1),X,R\} \\
&\vdots \\
\delta\{Q(i,s-2),B\} &= \{Q(i,s-1),X,R\} \\
\delta\{Q(i,s-1),B\} &= \{Q(i,s),Y,L\}
\end{aligned}$$

or generally:

$$\begin{aligned}
\delta\{Q(i,\mu),B\} &= \{Q(i,\mu+1),X,R\} & 0 \leq \mu \leq s-2 \\
\delta\{Q(i,s-1),B\} &= \{Q(i,s),Y,L\}
\end{aligned}$$

From now on TM starts to transfer the characters of the string s squares to the right. This work will be done character by character. Thus the first character to the left of the "X's" will be transferred to the right, and placed in the square with a "Y" (last square before the blanks). Then a "Y" will be printed on the last square bearing an "X" to the right, and the same process will be repeated until the s "X's" are successively located to the right of T_i .

So we include the moves:

$$\delta\{Q(i,s),X\} = \{Q(i,s),X,L\} \quad \begin{array}{l} \text{the head goes left as long} \\ \text{as the character read is "X".} \end{array}$$

$$\begin{aligned}
\delta\{Q(i,s),a_1\} &= \{Q(i,s+1),X,R\} \\
&\vdots \\
\delta\{Q(i,s),a_K\} &= \{Q(i,s+K),X,R\}
\end{aligned} \quad \text{set 1}$$

$$\begin{aligned}
\delta\{Q(i,s+1),X\} &= \{Q(i,s+1),X,R\} \\
&\vdots \\
\delta\{Q(i,s+K),X\} &= \{Q(i,s+K),X,R\}
\end{aligned} \quad \text{set 2}$$

$$\begin{aligned}
\delta\{Q(i, s+1), Y\} &= \{Q(i, s+K+1), a_1, L\} \\
&\vdots \\
\delta\{Q(i, s+K), Y\} &= \{Q(i, s+K+1), a_K, L\}
\end{aligned}
\quad \text{set 3}$$

or generally:

$$\delta\{Q(i, s), a_\lambda\} = \{Q(i, s+\lambda), X, R\} \quad \text{set 1}$$

$$\delta\{Q(i, s+\lambda), X\} = \{Q(i, s+\lambda), X, R\} \quad \begin{matrix} a_\lambda \in A \\ \lambda = 1, 2, \dots, K \end{matrix} \quad \text{set 2}$$

$$\delta\{Q(i, s+\lambda), Y\} = \{Q(i, s+K+1), a_\lambda, L\} \quad \text{set 3}$$

$$\delta\{Q(i, s+K+1), X\} = \{Q(i, s), Y, L\}$$

and finally:

$$\delta\{Q(i, s), T_i\} = \{Q(i, s+K+2), T_i, R\}$$

The next moves are the same as the moves in case B, after the execution of the routine "Expansion".

Actually, in case B, the next following state of TM, after the execution of "Expansion", was symbolized by $Q(i, r')$.

Thus for $|LHS_i| = 0$ it is: $r' = s+K+2$

In case B after this point all following states have as second subscript a function of r' . Thus no confusion arises if $r' = s+K+2$ and all the next states are defined for the case $|LHS_i| = 0$.

4.10 Values of subscript w

As we said before the subscript w is used for some states of TM, $Q(i, w)$, $Q(i, w+1)$ and $Q(i, w+2)$ which will have the following values for each case of an MA rule.

A. For $|LHS_i| = |RHS_i|$

$$(1) \quad \text{if } |LHS_i| = 0 \rightarrow w = 0$$

$$(2) \quad \text{if } |LHS_i| > 0 \rightarrow w = 2r-1$$

but because $r=s$ $w = r+(s-1)$

B. For $|LHS_i| < |RHS_i|$

$$(1) \quad \text{if } |LHS_i| = 0 \rightarrow w = K+2(s+1) \quad \text{Special case. Here } |LHS_i| > |RHS_i| = 0; \quad w \text{ depends on } K, \text{ in this case because of the different construction of TM moves.}$$

$$(2) \quad \text{if } |LHS_i| = 1 \rightarrow w = s+1$$

$$(3) \quad \text{if } |LHS_i| > 1 \rightarrow w = r+s+1$$

C. For $|LHS_i| > |RHS_i|$

$$(1) \quad \text{if } |LHS_i| = 1 \rightarrow w = 1$$

$$(2) \quad \text{if } |RHS_i| > 1 \rightarrow w = r+s+1$$

The above values of w are easily extracted (work which has already been done for each particular case). It should be noted here that $Q(i,w)$ is the state of TM after the application of the rule i and before a T_i has been scanned by the head of the machine.

5. SOME IMPORTANT REMARKS FOR THE TM's CONSTRUCTION

The construction of TM has now been completed. It remains to be proved that the constructed TM is able to do exactly what MA can do and no more. This means of course that it must also be proved that MA is able to do what TM can do.

Before we proceed with the above proofs we shall make some remarks about the constructed TM.

Our TM has been constructed so that:

- A. When the search for the applicability of a rule i starts, TM will be in state $Q(i,0)$ with its head scanning the square bearing the first character of the string.
- B. After the search for the applicability of a rule:
 1. If the rule was found inapplicable, the head of TM will be driven, by a suitable set of moves, to the square of the tape which is to the left of the square containing the first character of the string.
 2. If the rule was found applicable, it will be applied and the head will be driven to the same square of the tape, as in case 1, but by a different set of moves.
- C. When TM is in any one of the states $Q(i,0)$, with i $0 \leq i \leq n$, and its head is scanning the square with the first character of the string, there will be no other characters in the string apart from characters belonging to alphabet A .
- D.
 1. TM cannot loop or terminate while the applicability of a rule is tested or when this rule is applied. In other words TM does not loop or terminate when it is "working" for a rule of MA . This is guaranteed because in any case there are suitable moves which will drive the head of TM to the square with the characteristic for this rule, symbol T_i , and will (simultaneously) change the states of TM so that TM will be in state $Q(i,0)$ after the special symbol T_i has been scanned.
 2. TM does not terminate when its head is scanning a square different from the one which contains the first character of the string. (This can easily be seen from the previous case).

- E. The first move of TM does not correspond to any action of MA, but is used in order to bring the head of TM to the second square of the tape where the first character of the string is at this time.
- F. The moves leading to the final state of TM, $Q(n+2,0)$ can only be executed if the special symbol T_i is read by the head of TM, while TM, obviously, is in a state which is capable of having as next one the final state $Q(n+2,0)$. The only possible termination for TM occurs when TM is in its final state.

The above features are guaranteed by the very construction of TM, and as indicated below.

6. TERMINATION OF TM

As has already been stated in our remarks concerning the construction of TM, TM can terminate only if it is in a final state. No other kind of termination is possible.

Thus there is no way for the head to jump out from the left end of the tape, because the markers "S" or " T_i " are a kind of a left barrier for it.

There is, also, always a next move for TM (providing TM is not in its final state), because by construction, there is a next move defined for every possible combination of "state-character". Of course there are combinations of "state-characters" for which no next move is defined. These combinations however can never occur for any string given as input to TM, by reason of the specific construction of TM.

The unique final state of TM, $Q(n+2,0)$ can be "reached" only through moves of some specific "state-character" combination; these are the following:

1. After the application of a terminating rule i .

In this case the head will be driven left² to the square where the characteristic symbol " T_i " for this rule is located. At this time TM will be in state $Q(i, w_i)$.¹

The last move of TM, when T_i is scanned will be:

$$\delta\{Q(i, w_i), T_i\} = \{Q(n+2, 0), S, R\} \text{ if } |LHS_i| < |RHS_i|$$

Thus TM will terminate with its head pointing to the square where the first character of the string is.

2. When the last rule ($i=n-1$) has been tested and found inapplicable. In this case TM will be in state $Q(n-1, w+1)$ while its head will be going left to the square where T_{n-1} is. The move of TM when this square is scanned by its head will be:

$$\delta\{Q(n-1), w_{n-1}+1, T_{n-1}\} = \{Q(n+2, 0), S, R\}$$

TM will therefore terminate in this case with its head pointing again to the square where the first character of the string is.

- 1 The values of w have been given above for every kind of rule i .
- 2 If for this rule we have $|LHS_i| > |RHS_i|$ then the head will be driven right and when a Y is read the characteristic symbol S will be printed on this square and the move will be

$$\delta\{Q(i, w), Y\} = \{Q(n+2, 0), S, R\}$$

THEOREM 1.1 There is a set of TM moves equivalent to every MA rule.

One of the MA rules, say rule i , is chosen at random. We also suppose that a string $w \in \Sigma^*$ is given as input to MA and, by reason of an exoteric intervention, MA starts with rule i (in case $i \neq 0$).

In a parallel manner, we suppose that the same string w is given as input to TM, which after an equivalent corresponding exoteric intervention, starts by being in state $Q(i,0)$ and with its head scanning the square of the tape which contains the first character of w . The tape at this time has the form: " $\# \dots \# T_i w BBB \dots$ " (where the number of squares to the left of T_i can be any integer, even zero; but we assume that this number is zero for the easier counting of the " $\#$ "'s which may be added). Then we shall prove that in each of the following pairs of statements, statements a and b are equivalent.

Pair A.

- a. MA found the rule i to be applicable.
- b. TM, after a number of executions of some of its moves, is in state $Q(i, r-1)$, where $r = |LHS_i|$ and the square scanned by its head contains the same character as the last one in LHS_i .

Pair B. (providing that rule i is applicable)

- a. After the rule i has been applied the string " w " will have been modified to " w' ".
- b. After the execution of one of the following moves:

$$\delta\{Q(i, w_i), T_i\} = \{Q(\theta, 0), T_\theta, R\} \quad \text{if } |LHS_i| \leq |RHS_i|$$

or

$$\delta\{Q(i, w'_i), Y\} = \{Q(\theta, 0), T_\theta, R\} \quad \text{if } |LHS_i| > |RHS_i|$$

(where $0 \leq \theta < n-1$ or $\theta = n+2$; the values of w and w' have been given before, as for all other cases),

the tape of TM will have the form:

$$T_0 w' BBB \dots \quad \text{if } |LHS_i| \leq |RHS_i| \quad \text{or}$$

$$\neq \neq \dots \neq T_0 w' BBB \dots \quad \text{if } |LHS_i| > |RHS_i|.$$

The head of TM will point to the square with the first character of "w'". (Obviously the strings "w'" in the two cases should not be the same, but we nevertheless use the same symbol w', since only one of the conditions can occur).

Pair C

- a. MA terminates after the application of the terminating rule i.
- b. TM halts after the execution of one of the moves:

$$\delta\{Q(i, w_i), T_i\} = \{Q(n+2, 0), S, R\} \quad \text{if } |LHS_i| \leq |RHS_i|$$

or

$$\delta\{Q(i, w'_i), Y\} = \{Q(n+2, 0), S, R\} \quad \text{if } |LHS_i| > |RHS_i|$$

Pair D

- a. After the application of the nonterminating rule i, the next rule tested is rule θ . (If MA is not labelled $\theta=0$ but $0 \leq \theta < n$ otherwise).
- b. After the execution of the move:

$$\delta\{Q(i, w), T_i\} = \{Q(\theta, 0), T_\theta, R\} \quad \text{if } |LHS_i| \leq |RHS_i|$$

or

$$\delta\{Q(i, w'), Y\} = \{Q(\theta, 0), T_\theta, R\} \quad \text{if } |LHS_i| > |RHS_i| ,$$

where $0 \leq \theta < n$ and $T_\theta \neq S$, the head of TM points to the square with the first character of the string "w'", TM being in state $Q(\theta, 0)$.

Pair E

- a. Rule i was found inapplicable by MA and the string w remains unmodified.
- b. At some time TM is in state $Q(i, w+1)$ while its head goes left (looking for a T_i). At this stage, the tape has the form which it had before the start of the applicability search (for this rule).

Pair F

- a. If the inapplicable rule i is not the last of MA, the rule $i+1$ is the next tested. Otherwise MA terminates.
- b. With TM in state $Q(i, w+1)$, at some time its head will be scanning the square with the symbol T_i . If then for the subscript i is : $i < n-1$

the move: $\delta\{Q(i, w+1), T_i\} = \{Q(i+1, 0), T_{i+1}, R\}$

will be executed.

But if $i = n-1$

the move: $\delta\{Q(i, w+1), T_i\} = \{Q(n+2, 0), S, R\}$

will be executed and TM will halt.

(In both cases after this move, TM will be in state $Q(i+1, 0)$ if $i < n-1$ or $Q(n+2, 0)$, if $i = n-1$, and the head will point to the first square to the right of the one with the symbol T_i .)

ProofsPair A

$a \rightarrow b$

The fact that rule i is applicable for MA, means that the LHS of this rule is really located somewhere in the string w . Of course it

is not impossible that there is more than one occurrence of LHS_i in w but our MA and TM are concerned only with the left most one.

Because the tape of TM has the form $T_i w BBB \dots$ this means that the LHS_i is located in some of the successive squares of the tape which contain " w ".

According to its construction, TM compares one by one the characters of LHS_i with the characters of the squares of the tape containing w , starting from the first character of w . Note that TM is in state $Q(i,0)$ as long as the first character in LHS_i has not been read yet in the tape. When this character has been read TM changes state to $Q(i,1)$ printing a marker " $s_{a_{i,1}}$ " on this square (if $|LHS_i| > 1$) and its head goes further right. If then after some time, p successive characters ($0 \leq p < |LHS_i|$) of LHS_i , (including the first) have been read in successive squares of the tape, TM will be in state $Q(i,p)$ and its head scanning the next, to the right, square of the tape. If now the next square in the tape does not contain the same character as the p th character in LHS_i , the head of TM will go left with TM in a state $Q(i,w+2)$ looking for the marker " $s_{a_{i,1}}$ ". Then the same process is repeated starting again with the first character in LHS and with the character which is in the next to the right square of the tape.

Since from (a) it is known that the LHS_i is in fact somewhere in the tape, it is also known that after some time TM will be in state $Q(i,r-1)$, which means that $r-1$ characters of LHS_i have been read in successive squares of the tape. At this time the head of TM will be scanning a square of the tape, which will be to the right of the $r-1$ successive squares containing all, but the last, characters of LHS_i in succession.

If the character contained in this square is not the same as the last in LHS_i , the head will go left again with TM in state

$Q(i, w+2)$ etc. But at some time TM will be in state $Q(i, r-1)$ and its head scanning the square with the same character as the last in LHS_i .

Q.E.D.

$b \rightarrow a$

The proof is analogous to $a \rightarrow b$.

Pair B

$a \rightarrow b$

Since rule i has been applied by MA, the LHS_i exists in w and thus it must be located in some successive squares in the tape of TM. From the pair (of Statements) A it is known that if this happens, then at some time TM will be in state $Q(i, r-1)$ and scanning the (next to the right) square which contains the same character as the last in LHS_i .

But as rule i has been chosen at random and it can be any rule, $0 \leq i < n$, we shall distinguish three different cases.

$$1. \quad |LHS_i| = |RHS_i|$$

In this case the substitution of LHS_i by RHS_i , which will take place, is straightforward. The head going left will be printing the characters of RHS_i , starting from the last, in the squares where the characters of LHS_i were. At the same time TM will be changing states, being in the state $Q(i, 2r-1)$ when the substitution has been completed. While TM will be in this state, its head will be going left looking for a " T_i " and leaving everything else unchanged. Thus when the square with the symbol " T_i " is scanned the only modification in the tape is that the left most occurrence of LHS_i has been substituted by RHS_i (that is, the same modification as in the case of MA when the rule i was applied). Hence if the modified

string of MA is w' , the successive squares of TM which contained w before, now contain w' . Finally, after the move:

$$\delta\{Q(i, 2r-1), T_i\} = \{Q(\theta, 0), T_\theta, R\}$$

where (in this case) $w=2r-1$,

the tape will have the form:

$$T_\theta w' BBB \dots$$

with the head of TM pointing to the square containing the first character of the string w' .

$$2. \quad \underline{|LHS_i| < |RHS_i|}$$

In this case, when the last character of LHS_i is in the square of the tape scanned by the head of TM, with TM in state $Q(i, r-1)$, a different kind of substitution is being started. All the successive characters in the tape constituting the LHS_i are substituted by X's. Then by the subroutine "Expansion", the tape is expanded by a number of squares $s-r$, (where $s = |RHS_i|$, $r = |LHS_i|$). These squares are filled with X's and placed to the right of the other squares containing X's.

Thus if $LHS_i = a_{i_1} a_{i_2} \dots a_{i_r}$, we assume that $w = w_1 a_{i_1} a_{i_2} \dots a_{i_r} w_2$ where $w_1, w_2 \in \Sigma^*$ (if one or more rules had been applied, $w_1, w_2 \in A^*$).

The tape will then have the form:

$$T_i w_1 a_{i_1} a_{i_2} \dots a_{i_r} w_2 BBB \dots$$

After the substitution of the characters of LHS_i by X's and the addition of the squares filled with X's, that is, $s-r$ new squares, the string will be:

$$w_X = w_1 XX \dots X w_2 \text{ where the number of squares with X's is } s.$$

The tape at this time will have the form:

$$T_i w_1 XX \dots X w_2 BBB \dots$$

TM, before it started the execution of subroutine "Expansion" was in state $Q(i,r)$, which is next to state $Q(i,r-1)$ according to the move:

$$\delta\{Q(i,r-1), a_{i,r}\} = \{Q(i,r), X, L\}$$

Thus after the execution of "Expansion" (for which TM uses a number of special states) and before the substitution of X's by the characters of RHS_i , TM is in state $Q(i,r')$, where

$$r'=1, \text{ if } r=1 \quad \text{or} \quad r'=r+1 \text{ if } r>1.$$

The X's now are substituted by the characters of RHS_i from right to left. After every substitution of an "X" by the corresponding character of RHS_i , TM changes state and when all X's (totally s) have been substituted by the characters of RHS_i , TM will be in state $Q(i,r'+s)$, with its head going left and looking for a T_i . At this time:

- a) the modified string will be

$$w_1 a_{j_1} a_{j_2} \dots a_{j_s} w_2 = w' ;$$

- b) the tape will have the form:

$$T_i w' BBB \dots$$

When the head will scan the square with the T_i the following move will be executed:

$$\delta\{Q(i,w), T_i\} = \{Q(\theta, 0), T_\theta, R\}$$

(Here $w=r'+s$)

Then the head will point to the square with the first character of the string w' , and the tape will have the form:

$$T_\theta w' BBB \dots$$

In case $|LHS_i| = 0$ the proof is analogous.

3. $|LHS_i| > |RHS_i|$

As in the case " $|LHS_i| = |RHS_i|$ ", in this case too, TM substitutes the characters of LHS_i by the ones of RHS_i starting with the last characters and moving from right to left. Obviously only s characters of LHS_i will be substituted by the characters of RHS_i . The remaining $r-s$ characters of LHS_i will be substituted by a Y followed by $r-s-1$ X 's (where $r = |LHS_i|$, $s = |RHS_i|$).

We assume now that the input string w is:

$$w = w_1 a_{i_1} a_{i_2} \dots a_{i_r} w_2,$$

where $a_{i_1} a_{i_2} \dots a_{i_r} = LHS_i$ and $w_1, w_2 \in \Sigma^*$.

(in case that one or more MA rules had been applied by the corresponding sets of TM moves, $w_1, w_2 \in A^*$).

Before any substitution the tape has the form:

$$T_i w_1 a_{i_1} a_{i_2} \dots a_{i_r} w_2 BBB \dots$$

(In fact when the LHS_i has been found in the tape and before any substitution of LHS_i characters by RHS_i ones, the character a_{i_1} has been substituted by the marker $S_{a_{i_1}}$. Since this marker is being used only in order to indicate the first character of LHS_i in the tape, we may assume that it is in effect the same a_{i_1} which it stands for).

After the substitution of the s last characters of LHS_i by the RHS_i ones where:

$$RHS_i = a_{j_1} a_{j_2} \dots a_{j_s},$$

the input string will have been modified to

$$w''' = w_1 a_{i_1} a_{i_2} \dots a_{i_{r-s}} a_{j_1} a_{j_2} \dots a_{j_s} w_2.$$

Similarly after the substitution of the remaining LHS_i characters

by one Y and $r-s-1$ X's, w'' will have been modified to

$$w_1'' = w_1 \text{ XX } \dots \text{ X Y } a_{j_1} a_{j_2} \dots a_{j_s} w_2$$

At the same time the tape will be like:

$$T_i w_1 \text{ XX } \dots \text{ X Y } a_{j_1} a_{j_2} \dots a_{j_s} w_2 \text{ BBB } \dots$$

When TM starts the substitution of LHS_i by RHS_i it is in state $Q(i, r-1)$. Thus after the substitution of the last s characters of LHS_i by the RHS_i ones, TM will be in state $Q(i, r+s-1)$, because it changes state after every character's substitution. Then a Y will be printed on the next square to the left and the state of TM will be $Q(i, r+s)$. During the substitution of the remaining characters of LHS_i by X's, TM will remain in the same state $Q(i, r+s)$. Thus when the last X has been printed on the square with the marker $S_{a_{i1}}$, TM will start the execution of the subroutine "Cut off" for which a number of special states is used.

After the "Cut off" execution, all X's and the one Y will be located in the left part of the modified string w_1'' which will now be:

$$w_2'' = \text{XX} \dots \text{X Y } w_1 a_{j_1} a_{j_2} \dots a_{j_s} w_2$$

At the same time the tape will have the form:

$$"/\text{ XX} \dots \text{X Y } w_1 a_{j_1} a_{j_2} \dots a_{j_s} w_2 \text{ BBB } \dots "$$

The unique (at the moment) $"/\text{}$ is printed on the square where the T_i was before.

With TM now in state $Q(i, r')$ and its head scanning the square with the first "X" (the one to the right of $"/\text{}$) the following move is executed:

$$\delta\{Q(i, r'), X\} = \{Q(i, r'), /\text{, R}\}$$

$$\text{where } r' = r+s+1 \quad \text{if } r > 1 \quad \text{or} \quad r' = r = 1$$

This move is successively repeated until all X's have been substituted by $\#$'s. After the substitution of Y by T_θ , according to the move:

$$\delta\{Q(i, w'), Y\} = \{Q(\theta, 0), T_\theta, R\}$$

where $w' = r'$, the tape will have the form:

$$"\# \# \dots \# T_\theta w_1 a_{j_1} a_{j_2} \dots a_{j_s} w_2 \text{ BBB } \dots";$$

but

$$w_1 a_{j_1} a_{j_2} \dots a_{j_s} w_2 = w'$$

hence the form of the tape will be:

$$"\# \# \dots \# T_\theta w' \text{ BBB } \dots"$$

Since rule i was the first applied (in TM's way), the number of squares with $\#$'s is equal to $r-s$. (If rule i was not the one first applied, $r-s$ squares with $\#$'s would be added to the already existing ones).

The head of TM, after the above last move, will point to the first square of the string w' , and TM will be in state $Q(\theta, 0)$.

Q.E.D.

$b \rightarrow a$

The proof is analogous.

Pair C

$a \rightarrow b$

From (a) we know that rule i is a terminating one. By the construction of TM, for every terminating MA rule i ($0 \leq i < n$) we include among the set of TM moves corresponding to the MA rule i, the following move:

$$\delta\{Q(i,z),\Omega\} = \{Q(n+2),0\},S,R\}$$

$$\begin{array}{llll} \text{where } z=w & \text{and } \Omega=T_i & \text{if } |LHS_i| \leq |RHS_i| \\ z=w' & \text{and } \Omega=Y & \text{if } |LHS_i| > |RHS_i| \end{array}$$

Because, from pair B, we know that the above move will be executed, after the execution, TM will halt, being in the final state $Q(n+2,0)$.

b→a The proof is similar.

Pair D

a→b

Among the TM moves corresponding to the i th MA rule, we have included the move

$$\delta\{Q(i,z),\Omega\} = \{Q(\theta,0),T_\theta,R\}$$

$$\begin{array}{llll} \text{where } z=w & \text{and } \Omega=T_i & \text{if } |LHS_i| \leq |RHS_i| \\ \text{or } z=w' & \text{and } \Omega=Y & \text{if } |LHS_i| > |RHS_i| \end{array}$$

Because rule i is not a terminating one we have:

$$0 \leq \theta < n \quad \text{and (obviously)} \quad T_\theta \neq S = T_{n+2}$$

Also, as it follows from pair B, that the above TM move will be executed, after the execution the head of TM will be scanning the square with the first character of the string and TM will be in state $Q(\theta,0)$.

b→a The proof is similar.

Pair E

a→b

From pair A we know that if and only if rule i is applicable, TM will be, at some time, in state $Q(i,r-1)$ with its head scanning a

square with the same character as the last in LHS_i . But this is not possible as, by (a), rule i is inapplicable.

The only way for TM to reach the state $Q(i, w+1)$ is, if TM is in state $Q(i, 0)$ and its head is scanning the first blank square of the tape. Naturally, TM may be in several states $Q(i, t)$ where $0 \leq t \leq r-1$ before that actually occurs. Whenever TM is in any one of the states $Q(i, t)$ where $1 \leq t \leq r-1$ and the next character scanned is not the same as the corresponding one in LHS_i , the following things happen:

- a. TM changes state to $Q(i, w+2)$ and the head goes left by t squares.
- b. On the t 'th square to the left (where the marker $S_{a_{i1}}$ is located at this stage) the first character of LHS_i , " a_{i1} " is printed. (This character was there originally).
- c. TM changes state to $Q(i, 0)$ and the head goes right starting the search again.

Thus, because TM will never in this case be in state $Q(i, r-1)$ with its head scanning a square with the same symbol as the last in LHS_i , we see that TM will fail to find the LHS_i on successive squares of its tape. Thus at some time, and while TM is in state $Q(i, 0)$, its head will be scanning the first blank square of the tape and the following move will be executed:

$$\delta\{Q(i, 0), B\} = \{Q(i, w+1), B, L\}$$

After this move, the head will be going left, looking for a T_i , while TM will be in the same state $Q(i, w+1)$.

Before the above move, the only changes effected on the tape were the substitutions of every character, which was the same as the first in LHS_i by the marker $S_{a_{i1}}$. But as has been explained, every time TM failed to find the whole LHS_i in successive squares of the tape

(and after t successive characters of LHS had been found on t successive squares of the tape, where $l \leq t \leq r-1$) it restored this first character of LHS_i on the square with the marker S_{ai_1} and so the tape remained unmodified.

$b \rightarrow a$ The proof is similar.

Pair F

$a \rightarrow b$

Since rule i is inapplicable, we know from pair E that at some time TM will be in state $Q(i, w+1)$ with its head going left and looking for a T_i . It is also known that for every rule i there is always a symbol T_i on a specific square of tape (actually on the square next to the left of the square carrying the first character of the string). While the head will be going left, it will be scanning squares with the characters of the string w . By the construction of TM, there are K moves:

$$\delta\{Q(i, w+1), \xi\} = \{Q(i, w+1), \xi, L\}$$

defined $\forall \xi \in A, \quad (A \equiv \{a_1, a_2, \dots, a_K\})$

It is also known from pair E, that all the squares lying between the one containing the symbol T_i and the first blank square of the tape, contain characters which belong to A , because nothing has changed on the tape, since the applicability search for rule i has started.

It follows from the above that at some time the head will scan the square with the symbol T_i , while TM will be in state $Q(i, w+1)$.

If now the rule i is not the last rule of MA, there will be a next rule to be tested and this will be the rule $i+1$. In this case, by the TM's construction, there will be the move

$$\delta\{Q(i, w+1), T_i\} = \{Q(i+1, 0), T_{i+1}, R\}$$

to be executed.

But if rule i is the last of MA, the above move will not exist (by construction). Instead there will be the move

$$\delta\{Q(i, w+1), T_i\} = \{Q(n+2, 0), S, R\},$$

after whose execution TM, being in the final state $Q(n+2, 0)$, will halt.

$b \rightarrow a$ The proof is similar.

7. EQUIVALENCE BETWEEN AN MA RULE AND A TM SET OF MOVES

After all these constructions, proofs and remarks, we are now entitled to claim that:

- a. For every MA rule i , where $0 \leq i < n$, there is a corresponding set of TM moves M_i , equivalent to it.
- b. For every set of TM moves M_i , $\forall i : 0 \leq i < n$, there is a corresponding MA rule i , equivalent to this set of moves.

The set of moves M_i ($\forall i : 0 \leq i < n$) are defined as follows:

$$M_i = (\Delta i \cup E_i \cup C_i)$$

where:

Δi is the set of all TM moves defined for all the pairs of "state-character" $\{Q(i, z_i), \xi\}$

with: z_i an integer such that $0 \leq z_i \leq w_i + 2$ and $\xi \in \Gamma$.

E_i and C_i :

It can be easily seen that, by the construction of "Expansion", the sets of TM moves created for the simulation of MA rules with the property $|LHS| < |RHS|$, may not be the same for every rule, but they are dependent on the difference $(|LHS| - |RHS|)$. Thus we assume that for the set of "Expansion" moves, created for simulation

of rule i , is E_i . This E_i will, of course, be empty if $|LHS_i| \geq |RHS_i|$.

For the simulation of all MA rules, for which $|LHS| > |RHS|$, the sets of subroutine "Cut off" moves created are not different. Thus we use C_i for the set of TM moves, constituting the subroutine "Cut off", created for the simulation of the MA rule i . C_i will have only two values for all MA rules:

$$\forall i : 0 \leq i < n, \text{ if } |LHS_i| > |RHS_i|,$$

C_i will be the set of "Cut off" moves: otherwise C_i will be empty.

Thus for E_i and C_i we have:

$$E_i = \begin{cases} \text{the set of "Expansion" moves created} & \text{if } |LHS_i| < |RHS_i| \\ \emptyset & \text{if } |LHS_i| \geq |RHS_i| \end{cases}$$

$$C_i = \begin{cases} \text{the set of "Cut off" moves} & \text{if } |LHS_i| > |RHS_i| \\ \emptyset & \text{if } |LHS_i| \leq |RHS_i| \end{cases}$$

THEOREM 1.2

A number σ of successive inapplicable rules, lying between two applicable ones (that is, we envisage a case in which one of the applicable rules, say rule i , has been applied, and the other, say i' , is due to be tested and applied after the σ inapplicable rules have been tested) leave a string exactly as it was after the application of the first of the two applicable rules for both MA and TM. (Obviously TM tests and applies the MA rules by its corresponding sets of moves). Also, after the applicability search for the σ inapplicable rules has been completed, and the applicability search

for rule i' is starting, TM will be in state $Q(i', 0)$ with its head scanning the first square to the right of the one with the symbol $T_{i'}$, and its tape will have the form

$$\# \# \# \dots \# \# T_{i'} w BBB' .$$

The only difference on the tape between its form immediately after the application of rule i , and its form just at the beginning of the applicability search for rule i' , is that whereas the characteristic symbol was $T_{i+\sigma_1}$ in the former case, it is $T_{i'}$ in the latter, where

$$\sigma_1 = \begin{cases} 0 & \text{if MA is unlabelled} \\ \sigma_i & \text{if MA is labelled} \end{cases}$$

with $\sigma_i : 0 \leq \sigma_i < n-1$.

Finally for σ it must be : $\sigma < n - \sigma_1 - 1$

Proof

For MA the result is obvious by definition. As for TM the proof is as follows:

By hypothesis, it is given that after the application of rule i , there will be tested σ inapplicable rules. The first one σ_1 is defined as follows:

$$\sigma_1 = \begin{cases} 0 & \text{if MA is unlabelled} \\ \sigma_i & \text{if MA is labelled, where } \sigma_i \text{ is the label of the} \\ & \text{next rule tested after rule } i \text{ was applied, and } \sigma_i \\ & \text{must satisfy } 0 \leq \sigma_i < n-1, \text{ because there is at least} \\ & \text{one applicable more rule (rule } i'). \end{cases}$$

Thus we assume that the σ inapplicable rules are:

$$\sigma_1, \sigma_2, \dots, \sigma_\sigma ; \quad \text{where:}$$

1 n is the number of MA rules, with the rule "zero" as first rule, and the rule " $n-1$ " as last.

$$\begin{aligned}
 \sigma_2 &= \sigma_1 + 1 \\
 \sigma_3 &= \sigma_2 + 1 \\
 &\vdots \\
 \sigma_\sigma &= \sigma_{(\sigma-1)} + 1
 \end{aligned}$$

and for j such as $2 \leq j \leq \sigma$ we have:

$$\sigma_j = \sigma_{j-1} + 1 \quad \text{or} \quad \sigma_j = \sigma_1 + (j-1) \quad (1)$$

as can be easily shown.

Since there is another applicable rule σ_σ must satisfy

$$\sigma_\sigma < n-1 \quad (2)$$

where n is the number of the MA rules, the first rule being rule "0" (zero).

If in (1) we give to j the value σ we will have

$$\sigma_\sigma = \sigma_1 + (\sigma-1) \quad (3)$$

From 2 and 3 we have $\sigma < n - \sigma_1$.

From the pairs of statements E and F of the theorem 1.1, we deduce that if the string on the tape of TM is at some time w , it will also be w after the applicability search of an inapplicable rule. Thus after the application of rule i , TM will be in state $Q(\sigma_1, 0)$ (with σ_1 defined as above) while its head will be scanning the first square to the right of the one with the character T_{σ_1} . The tape will have then the form

$$\# \# \# \dots \# \# T_{\sigma_1} w \text{ BBB } \dots$$

at this time.

After the applicability search, by TM, for the inapplicable rule σ_1 , its state will be $Q(\sigma_2, 0)$ with its head scanning the first square to the right of the one with a T_{σ_2} . The only difference in the tape (which will now have the form:

$$\# \# \# \dots \# \# T_{\sigma_2} w \text{ BBB } \dots ,)$$

will be the substitution of the character T_{σ_1} by T_{σ_2} .

The same thing will happen after the applicability search for the next inapplicable rule σ_2 . That is, there will be no modification on the tape, apart from the substitution of the characteristic symbol T_{σ_2} by T_{σ_3} . The head of TM will be also scanning at this time the same square of the tape as in the former cases (the first square to the right of the one with the character T_{σ_3}). Finally TM will be in state $Q(\sigma_3, 0)$ and starting the applicability search for the next inapplicable rule, σ_3 .

It can very easily be seen (it may be trivially proved by induction) that $\forall j : 2 \leq j \leq \sigma$ and $\sigma < n - \sigma_1$ (where j is the number of successive inapplicable rules), the same things will happen. Thus, after the applicability search of the σ th successive inapplicable rule, the only modification of the tape will be the substitution of the characteristic symbol T_{σ} by $T_{\sigma_{\sigma+1}}$, where $\sigma_{\sigma+1} = i'$, by our hypothesis; so $T_{\sigma_{\sigma+1}} = T_{i'}$. TM will now be in state $Q(\sigma_{\sigma+1}, 0) = Q(i', 0)$ and starting the applicability search for the rule i' ($i' = \sigma_{\sigma+1}$), with its head scanning the first square to the right of the one with the character $T_{i'}$. Thus the tape will have the form

$"\# \# \# \dots \# T_{i'} w BBB"$

and the string will be the same as it was after the application of the preceding applicable rule i , that is " w ".

In proving this theorem we can easily see that we have also proved:

THEOREM 1.3

For MA and TM, a number σ of successive inapplicable rules lying between two other rules i and i' does not effect any other modification in the string w of MA or the tape of TM as they were immediately after the applicability search for rule i (and its

application, if it was applicable) except for the modification of the characteristic symbol T_i . After the applicability search for σ successive inapplicable rules, TM will be in state $Q(i',0)$, (if it was in state $Q(i+1,0)$ after the applicability search for rule i) with its head scanning the first square to the right of the one with the characteristic symbol $T_{i'}$.

8. TWO COMMENTS

1. As must have been already noticed, we often use the expression: "the first square to the right of the one with the character T_j " ($0 \leq j < n$), where T_j is the character printed on the square which is to the left of the square with the first character of the string, or to the left of the first blank square if it happens that the string on the tape is empty. We use this expression which is always true, independently of whether or not the string on the tape is empty.

An expression equivalent to the above when the string on the tape is not empty is the following: "the square with the first character of the string".

2. As is given, our MA rule has n rules and its first rule is rule zero (which means, of course, simply that the first rule is named rule "zero").

Similarly

the second rule is rule one
the third rule is rule two
.
.
.
and finally the nth rule is rule $n-1$

Thus, whenever we talk about the "rule i " (that is the rule named rule " i ") of MA, we mean the $(i+1)$ th rule, and vice versa.

THEOREM 1.4

The string $w \in \Sigma^*$ is given as input to both MA and TM. Then the following two statements are equivalent:

(a) After the application of p rules q_1, q_2, \dots, q_p where q_1, q_2, \dots, q_{p-1} are not terminating and where we symbolize by q_i ($1 \leq i \leq p$) the i th rule applied since MA has started, p satisfying the condition $1 \leq p < \infty$, the input string " w " will have been transformed to $w_{q_p} \in A^*$, and the next rule tested will be the rule θ_{q_p} .

(If MA is labelled: $0 \leq \theta_j < n$. In case MA is unlabelled, $\theta_j = 0$; where θ_j stands for the rule tested immediately after rule j was applied, with $j : 0 \leq j < n$ and n the number of MA rules).

(b) After the execution by TM, since its starting point, of p moves of the type

$$1 : \delta\{Q(q_i, w_{q_i}), \Omega_{q_i}\} = \{Q(\theta_{q_i}, 0), T_{\theta_{q_i}}, R\} \quad 1$$

with $\theta_{q_i} \neq n+2, \forall i < p$, the tape of TM will have the form:

$$" \# \# \dots \# T_{\theta_{q_p}} w_{q_p} BBB \dots " \quad 2$$

and TM will be in state $Q(\theta_{q_p}, 0)$ with its head scanning the first square to the right of the one with the character $T_{\theta_{q_p}}$.

1 and 2 The w_j, Ω_j are defined as follows:

- a. w_j : Its values depend on j and have been given before for all possible cases.
 - b. Ω_j : if $|LHS_j| \leq |RHS_j|$ then $\Omega_j = T_j$ and $\Omega_j = Y$ otherwise.
2. The number of squares to the left of $T_{\theta_{q_p}}$ will be zero if no MA rule of the type $|LHS_i| > |RHS_i|$ with $1 \leq i \leq p$, is included among the rules q_1, q_2, \dots, q_p .

Proof by induction

a→b

For $i=1$ we have, from our hypothesis, that the only applied rule is rule q_1 . It is possible that a number of rules were tested and found inapplicable, before the start of the applicability search for rule q_1 . In such a case MA and TM do not modify the string w and the tape respectively, according to theorems 1.2 and 1.3. TM is, also, in state $Q(q_1, 0)$ when it starts the applicability search for the rule q_1 , with its head scanning the first square to the right of the one with the character T_{q_1} .

In the case of MA, after the application of rule q_1 , the string $w \in \Sigma^*$ will have been transformed to $w_{q_1} \in A^*$; and since rule q_1 is the last applied, the next rule tested will be rule θ_{q_1} , where θ_{q_1} will be zero if MA is unlabelled and $0 \leq \theta_{q_1} \leq n-1$ otherwise.

From the pairs of statements A and B of the theorem 1.1 we have that TM, after the applicability search for rule q_1 , will apply the rule with its set of moves corresponding to rule q_1 of the MA executing finally the move:

$$\delta\{Q(q_1, w_{q_1}), \Omega_{q_1}\} = \{Q(\theta_{q_1}, 0), T_{\theta_{q_1}}, R\}$$

which will be the only one of type 1 executed up to this time.

If we suppose that there were more moves of this type executed before the execution of the said move, then, by the pair B of statements of the theorem 1.1 we have that additional rules should have been applied before rule q_1 was tested. But this, naturally, could not happen, as it would contradict our hypothesis.

According to the same pair of statements of the theorem 1.1, the tape of TM will have the following form after the execution of the above described move.

$$"\# \# \dots \# T_{\theta_{q_1}} w_{q_1} BBB \dots"$$

where the number of squares to the left of the square with the character $T_{\theta_{q_1}}$ will be zero if $|LHS_{q_1}| \leq |RHS_{q_1}|$. The head of TM, also, will point to the first square to the right of the one with the character $T_{\theta_{q_1}}$. Consequently we have proved that for $i=1$ (a) implies (b).

We assume now that (a) implies (b) if $i=p-1$, that is in case that $p-1$ nonterminating rules have been applied, and we shall prove that (a) implies (b) if the number of applied nonterminating rules is p .

From our inductive hypothesis we have that in MA $p-1$ nonterminating rules have been applied, and thus that $p-1$ TM moves of the type 1 have been executed by TM. Since the $(p-1)$ th applied rule is rule q_{p-1} , the last of these moves in TM will be:

$$\delta\{Q(q_{p-1}, 0), \Omega_{q_{p-1}}\} = \{Q(\theta_{q_{p-1}}, 0), T_{\theta_{q_{p-1}}}, R\}.$$

The tape after the execution of this move will have the form:

$$"\# \# \dots \# T_{\theta_{q_{p-1}}} w_{q_{p-1}} BBB \dots"$$

and TM will be in state $Q(\theta_{q_{p-1}}, 0)$ with its head scanning the first square to the right of the one with the character $T_{\theta_{q_{p-1}}}$.

(We open a parenthesis here in order to explain the meaning of " q_{p-1} " which is being used as the name of the $(p-1)$ th applied rule; p has been defined to be: $1 \leq p < \infty$. On the other hand we know that for q_i (where q_i is the integer representing the actual name of the i th applicable rule) we must have $0 \leq q_i < n, \forall i: 0 \leq i \leq p$. This of course does not mean that $p < n$. On the contrary, p_i can be a number much greater than n . This can be readily explained

by the fact that for some pairs p_i, p_j , where $i \neq j$, it is possible that $q_{p_i} = q_{p_j}$.

By (a), there will remain another applicable rule after the application of the $(p-1)$ th applied rule, the additional rule becoming the p th applied rule after its application. This rule is named rule q_p . It is possible that after the application of the rule q_{p-1} , a number of inapplicable rules will be tested before the applicability search for rule q_p starts. According to the theorem 1.2 there will be no modifications on the tape of TM (or the string of MA) apart from several possible substitutions of the characteristic symbols T_j .

Thus if there were σ successive inapplicable rules, tested after the application of the $(p-1)$ th applied rule (which is rule q_{p-1}) and before the applicability search of the p th applicable rule, (that is before the applicability search for rule q_p), the following should hold good according to theorem 1.2.

With the completion of the applicability search for the σ successive inapplicable rules, and when the applicability search for rule q_p is starting, TM will be in the state $Q(q_p, 0)$ with its head scanning the first square to the right of the one with the symbol T_{q_p} , and its tape having the form:

$$" \neq \neq \dots \neq T_{q_p} w_{q_{p-1}} BBB " .$$

The only difference on the tape between its form as it was after the application of rule q_{p-1} , and as it is at the beginning of the applicability search for rule q_p , is that the characteristic symbol $T_{q_{p-1}}$ has been substituted by T_{q_p} .

Since we know from (a) that rule q_p (the p th to be applied) is applicable, it follows (according to the pairs of statements A and B of theorem 1.1), that TM after applying the rule q_p , by its

corresponding set of moves, will finally execute the move:

$$\delta\{Q(q_p, w_{q_p}), \Omega_{q_p}\} = \{Q(\theta_{q_p}, 0), T_{\theta_{q_p}}, R\}.$$

which is of the type 1.

Just as in the previous case, TM after the execution of the above move will be similarly in state $Q(\theta_{q_p}, 0)$ with its head scanning the first square to the right of the one with the character

$T_{\theta_{q_p}}$, while the tape will have the form:

$$" \# \# \dots \# T_{\theta_{q_p}} w_{q_p} BBB \dots "$$

Q.E.D.

$b \rightarrow a$

The proof is omitted as being analogous to that of $a \rightarrow b$.

THEOREM 1.5

A string, suppose $w \in \Sigma$, is given as input to both MA and TM.

Then the following two statements are equivalent.

a: MA terminates after the application of p rules, where q_p is the last rule applied, and $w_{q_p} \in A^*$ is the form of the string after the termination of all operations in MA, $\forall 0 \leq p < \infty$.

b: TM executes p moves of the type

$$\delta\{Q(q_i, w_{q_i}), \Omega_{q_i}\} = \{Q(\theta_{q_i}, 0), T_{\theta_{q_i}}, R\} \quad (1)$$

and

b_1 : halts immediately after the p th execution;
[here $1 \leq p$]

or

b_2 : halts after the applicability search for σ successive inapplicable rules, where $\sigma = n - \theta_{q_p}$ and $0 \leq p$.

In both cases b_1 and b_2 , TM halts being in its final state $Q(n+2,0)$ with its head pointing to the first square to the right of the one with the character T_{n+2} , (where $\theta_{q_p} = n+2$ for case b_1). The tape after halting will have the form:

$$" \# \# \dots \# T_{n+2} w_{q_p} BBB \dots "$$

Proof

$a \rightarrow b$

b_1 . We assume that q_p is a terminating rule of MA. Obviously in this case $p \geq 1$. From (a) we know that MA has made p applications of its rules on the input string. From theorem 1.4 we have that TM has made p executions of type 1 moves, and that after the last of these moves is found in state $Q(\theta_{q_p}, 0)$ with its head scanning the first square to the right of the one with the character $T_{\theta_{q_p}}$. The tape of TM has at this time the form:

$$" \# \# \dots \# T_{\theta_{q_p}} w_{q_p} BBB "$$

and the last move of type 1 is:

$$\delta\{Q(q_p, w_{q_p}), \Omega_{q_p}\} = \{Q(\theta_{q_p}, 0), T_{\theta_{q_p}}, R\} \quad (2)$$

On the other hand, from our assumption that rule q_p is a terminating rule and from the pair C (of statements) of theorem 1.1 it follows that TM will halt after the execution of the move:

$$\delta\{Q(q_p, w_{q_p}), \Omega_{q_p}\} = \{Q(n+2, 0), S, R\} \quad (3)$$

Since only one move of type 1 has been constructed for each rule, we have that $\theta_{q_p} = n+2$. ($T_{n+2} = S$ as we have stated before). Thus TM will halt in this case immediately after the execution of move (2) and its tape will have finally the form:

$$" \# \# \dots \# T_{\theta_{q_p}} w_{q_p} BBB "$$

or, since $T_{\theta_{q_p}} = T_{n+2} = S$, \leadsto " $\neq \neq \dots \neq S w_{q_p} BBB \dots$ "

b_2 . Since rule q_p ($0 \leq p$) is not a terminating rule, a number of successive inapplicable rules will be tested after the application of this rule and before the termination of MA. (In case that $p=0$ it is easy to see that $w_{q_0} = w$). Thus it follows that MA will not terminate after the application of a terminating rule. This can only mean that MA will terminate after it has tested and found inapplicable its last rule. Thus since the last applicable rule of MA is rule q_p (or there is no applicable rule if $p=0$) and θ_{q_p} is the next rule to be tested, rule " $\theta_{q_p} + (\sigma-1)$ " must be the last rule of MA. (If MA is unlabelled $\theta_{q_p} = 0$; otherwise $0 \leq \theta_{q_p} < n$). It follows that $\theta_{q_p} + (\sigma-1) = n-1$ and $\sigma = n - \theta_{q_p}$.

From the pairs of statements D, E, F of theorem 1.1 we have that TM after the execution of its last type 1 move, will find inapplicable (according to its own proper way) every rule found inapplicable by MA. As is known, both MA and TM test as next rule the same rule, " $i+1$ " after the conclusion of the applicability search for an inapplicable rule i , $\forall i: 0 \leq i < n-1$. Thus in the case that rule θ_{q_p} is the next rule tested after the application of rule q_p , when the applicability search for the σ th successive inapplicable rule starts, according to theorems 1.2 and 1.3, it follows that:

TM will be in the state $Q(t, 0)$,
with its head scanning the first square to the right of the one with the character T_t ; the tape will at this time have the form:

$\neq \neq \dots \neq T_t w_{q_p} BBB \dots$

where $t = \theta_{q_p} + \sigma - 1$.

Since $\sigma = n - \theta_{q_p}$, we have $t = n - 1$. This means that TM will start the applicability search for the rule $n-1$, which is the last rule.

This rule is inapplicable for MA and from theorem 1.1, pairs E and F, we obtain that after the applicability search for this rule, TM will halt with the move:

$$\delta\{Q(n-1,0), T_{n-1}\} = \{Q(n+2,0), S, R\}$$

being in its final state $Q(n+2,0)$ and with its head pointing to the square on the right of the one with the character T_{n-1} . The tape, by theorems 1.2 and 1.3, will have the same form as after the execution of the last type 1 move (if $p=0$ $w = w_{q_0}$), that is

$$"\neq \neq \dots \neq T_{n+2} w_{q_p} BBB \dots "$$

Q. E. D.

$b \rightarrow a$

The proof is omitted as analogous to that of $a \rightarrow b$.

9. CONCLUSION

After our construction and proofs we know that TM is able to do exactly what MA can do and no more.

Thus if we consider TM and MA as recognising devices, they will behave and decide in exactly the same way for any string given as input to both.

TM has been constructed to behave as any Markov Algorithm MA. Thus TM will accept a string if it halts and reject if it loops. This happens because by its construction, it is not possible to halt for a non defined state-character pair (such a situation will never arise) and so it can only halt being in its unique final state.

CHAPTER II

FOR EVERY TURING MACHINE THERE IS AN
EQUIVALENT MARKOV ALGORITHM

As in chapter I, the proposition enunciated in the title of this chapter will be proved by the construction of a Markov Algorithm, "MA" for an arbitrary Turing Machine "TM".

1. DEFINITIONS OF TM AND MA

Suppose that a Turing Machine TM is defined as:

$$TM = (\bar{K}, \Sigma, \Gamma, \delta, q_0, F)$$

where \bar{K} : is the set of states of TM

Σ : is the set of input symbols finite sets

Γ : the set of all symbols used by TM

δ : a mapping from $\bar{K} \times \Gamma \rightarrow \bar{K} \times (\Gamma - \{B\}) \times \{L, R\}$

where B is the blank symbol which can
only be read and never typed.

q_0 : is the initial state of TM

$F \subseteq \bar{K}$: the set of final states.

We will construct a Markov Algorithm MA equivalent to TM.

MA will then be defined as follows:

$$MA = (A, \Sigma, R, N_p)$$

where $A = (\Gamma \cup (K - F) \cup \{P\} \cup \{C\})$ is the finite alphabet of MA;

$\Sigma \subseteq A$ is the set of input symbols;

P is a finite set of sequential rules of the type:

$LHS_i \rightarrow RHS_i, \forall i : 0 \leq i \leq p+5; (p \text{ is the number of defined TM moves});$

N_p is a set of instructions for the next rule tested.

We want MA to simulate exactly what TM does when a string w given to TM as inputs is given to MA as well.

2. SIMULATION OF TM'S STATES AND HEAD-MOVEMENTS BY MA.

When TM "is working" on the input string w , it is precisely defined at any stage of the process what will be done next. Or better, after every move of TM, the next move (if there is any) is unambiguously defined. The reason for this is that when the machine is in a state q the head reads a specific character " α " in one of the squares of the tape, and therefore it will be, by the use of

function δ :

$$\delta(q, \alpha) = \{p, \beta, d\}$$

where $\alpha \in \Gamma$; $\beta \in (\Gamma - \{B\})$; $q, p \in \bar{K}$; $d \in \{L, R\}$;

If $\delta(q, \alpha)$ is not defined, there is no next move for TM. Since we now want MA to simulate TM we will use markers q_i where $q_i \in (\bar{K} - F)$ (one marker for each non-final state of TM). At any particular stage during the simulation of TM by MA, if the head of TM is scanning the square i (i being the square which is i number of squares away from the beginning of the tape; the first square has consequently a distance 0), while TM is in state q , the marker q for MA would be at the first position to the left of the character (on the MA string) corresponding to this square (of the TM tape).

This is achieved initially by including among the MA rules, some rules applicable very early which will modify the input string " w " to " $q_0 w$ ", where q_0 is the initial state of TM.

Then at any stage during the simulation of TM by MA, a marker $q_i \in (\bar{K} - F)$ (the only one marker of this kind actually in the string at this stage) will specify:

- a. The state of TM (in which state TM would be)
- b. The position of the head of TM (where the head of TM would be).

So we see that MA has exactly the same kind of information as TM. TM uses its δ function for defining its next move, and, for this δ function, MA will use its corresponding rules. Thus for every TM move there will be a corresponding MA rule and vice versa.

3. CORRESPONDING MA RULES TO TM MOVES.

The only possible types of moves for TM are:

1. $\delta(q_i, \alpha) = (p_i, \beta, R)$
2. $\delta(q_j, \alpha) = (p_j, \beta, L)$ if the tape of TM is infinite to the right
alone α cannot be the first character of
the tape.

$$3. \delta(q_i^!, B) = (p_i^!, \gamma, R)$$

$$4. \delta(q_j^!, B) = (p_j^!, \gamma, L)$$

where $\alpha, \beta, \gamma \in (\Gamma - \{B\})$,

$$q_i, q_j, q_i^!, q_j^!, p_i, p_j, p_i^!, p_j^! \in \bar{K},$$

$p_i, p_j, p_i^!, p_j^! \notin F$, $q_i, q_j, q_i^!, q_j^!$ obviously do not belong to F because they have next states;

$B \in \Gamma$ is the blank symbol.

For these four types of TM moves, we can construct "equivalent" (or corresponding) MA rules:

$$1'. q_i \alpha \rightarrow \beta p_i$$

$$2'. \sigma q_j \alpha \rightarrow p_j \sigma \beta$$

$$3'. q_i^! B \rightarrow \gamma p_i^! B$$

$$4'. \sigma q_j^! B \rightarrow p_j \sigma \gamma B$$

block A

where $\alpha, \beta, \gamma \in (\Gamma - \{B\})$
 $\sigma \in \Gamma$

In the case in which $p_i, p_j, p_i^!, p_j^! \in F$, the corresponding MA rules will be:

$$1''. q_i \alpha \rightarrow \beta P$$

$$2''. \sigma q_j \alpha \rightarrow P \sigma \beta$$

$$3''. q_i^! B \rightarrow \gamma P B$$

$$4''. \sigma q_j^! B \rightarrow P \sigma \gamma B$$

block B

$\sigma, \alpha, \beta, \gamma \in (\Gamma - \{B\})$

4. CONSTRUCTION OF MA

Apart from the rules of MA which exactly correspond to TM moves, some more MA rules will have to be used on the one hand, for setting up initially the string for MA and on the other hand rules for the detailed carrying out of the simulation of TM. If for TM the input is $w \in \Sigma^*$, then for MA the input will become $q_0 w B$ after a number of applications of the initial rule. After the string is transformed to $q_0 w B$ none of these initial rules will be again applicable.

We are now in a position to present the complete definition of MA:

- 0 : $B'\xi \rightarrow \xi B'$ initial rules $\forall \xi \in \Sigma^1$
- 1 : $B' \rightarrow B$ the string now is $q_0 wB$
- . n rules corresponding to n TM moves not
- . leading to final states
- . block A
- . k rules corresponding to k TM moves leading
- . to final states
- . block B
- $n+K+2$: $P \rightarrow \Lambda$ the only terminating rule of MA
- $n+K+3$: $B \rightarrow C$ whenever this rule is applied, the next becomes
- $n+K+4$: $C \rightarrow C$ applicable as well, and in such a case MA will
- never terminate since it will be applying con-
- tinuously this rule, for ever.
- $n+K+5$: $\Lambda \rightarrow q_0 B'$ initial rule, the first applicable one.

1 The character ξ , used in rule "0", is a "generic" variable, for the elements of Σ . MA here is not supposed to be a labelled Markov Algorithm and so after the application of any one of its rules, the next rule tested is always rule "0". Thus before rule 1 becomes applicable there should always be some character $\alpha \in \Sigma$ to the right of B' . After B' has been moved to the right end of the string, the next rule applicable is rule 1.

If for some reason (e.g. because we desire MA to be as simple as it can be) we do not want to use generic variables, then supposing that Σ has m elements $\Sigma = \{a_0, a_1, \dots, a_{m-1}\}$, the following rules should be used instead of rule "0".

- 0_0 : $B'\alpha_0 \rightarrow \alpha_0 B'$
- 0_1 : $B'\alpha_1 \rightarrow \alpha_1 B'$
- .
- .
- 0_{m-1} : $B'\alpha_{m-1} \rightarrow \alpha_{m-1} B'$

The above block of rules is equivalent to rule "0" because there is only one B' in the string at any time and there can be no confusion in the applicability order of the MA rules concerned even if the generic variable ξ is used.

5. A SYMBOLIZATION FOR TM

If the string $w \in \Sigma^*$ is the input string to TM and the initial state of TM is q_0 and the first character of the string is σ_0 , the first move of TM will be:

$$\delta(q_0, \sigma_0) = (q_0, (\sigma_0), \sigma_0, (q_0), d)$$

where:

$q_0, (\sigma_0)$: is the new state of TM after the character σ_0 was scanned when TM was in state q_0 .

$\sigma_0, (q_0)$: the symbol printed in the square where σ_0 was before, TM being then in state q_0 .

d : has the value R, as this was the first move of TM.

(If however the tape is infinite in both directions then $d \in \{L, R\}$).

We will use $q_0, (\sigma_0, \sigma_1, \dots, \sigma_{p-1})$ to symbolize the state of TM, after its head has scanned p characters and it has made p moves since it started in state q_0 . Some of the scanned characters, $\sigma_0, \sigma_1, \dots, \sigma_{p-1}$, or even all, may be the same. So if Γ has for example n elements and for some integer p it happens that $p > n+1$ this only can mean that there are repetitions of the same characters among the characters σ_i , $0 \leq i \leq p-1$. The subscripts i of σ_i then have a double meaning. First each σ_i specifies a character in Γ and second, i shows that this character has been read by TM in its $i+1$ move without excluding of course the case, that the same character has been also scanned once or more before by TM.

So there may be $i, j : \sigma_i = \sigma_j$ with $i \neq j$ and $0 \leq i \leq p-1$, $0 \leq j \leq p-1$.

By $\sigma_p, [q_0, (\sigma_0, \dots, \sigma_{p-1})]$ we symbolize the new character printed in the square where σ_p was before, when TM was in state $q_0, (\sigma_0, \dots, \sigma_{p-1})$. Finally the $(p-1)$ th move of TM will be:

$$\delta(q_0, (\sigma_0, \sigma_1, \dots, \sigma_{p-1}), \sigma_p) = (q_0, (\sigma_0, \dots, \sigma_{p-1}, \sigma_p), [q_0, (\sigma_0, \dots, \sigma_{p-1})], d)$$

where $d \in \{L, R\}$.

THEOREM 2.1

If TM and MA are given the same string $w \in \Sigma^*$ as input, then the following statements are equivalent:

(A) The tape of TM, after $K+1$ moves (not leading to final states) have been executed, will have the form:

$$w'_{K+1} \sigma_{K+1} w_{K+1} BBB \dots$$

with TM in state q and its head scanning the character σ_{K+1} having distance r from the beginning of the tape.

(B) After $K+1$ applications of MA rules which correspond to TM moves (none leading to final states) the string of MA will have the form

$$w'_{K+1} q \sigma_{K+1} w_{K+1} B$$

with $|w'_{K+1}| = r$.

where: $w'_{K+1}, w_{K+1} \in (\Gamma - \{B\})^*$

$$\sigma_{K+1} \in \Gamma$$

$q \in K$, $B \in \Gamma$ is the blank symbol

K a finite integer, possibly zero

Also for a string X , $|X|$ denotes the length of X

(The distance of the first character from the beginning of the tape is 0).

Proof

From (A) \rightarrow (B) by induction

When TM starts, it is in state q_0 and is scanning the character in the first square of the tape, i.e. the one which has distance "0". The tape in the beginning has the form

$$\sigma_0 w_0 BBB \dots$$

$$\text{where: } \sigma_0 \in (\Sigma \cup \{\dot{\epsilon}\}), \quad w_0 \in \Sigma^* \quad \text{and} \quad \sigma_0 w_0 = w. \quad (1)$$

The first move of TM then is defined to be:

$$\delta(q_0, \sigma_0) = (q_0, (\sigma_0), \sigma_0, (q_0), d) \quad \text{where } d = R \quad (m1)$$

After the first move, TM is in state $q_0, (\sigma_0)$ and the head points to the second square of the tape. The tape now is modified to the form:

$$\sigma_0, (q_0) \sigma_1 w_1 BBB \dots \quad (2)$$

$$\text{where: } \sigma_1 \in (\Sigma \cup \{\dot{\epsilon}\}), \quad w_1 \in \Sigma^* \quad \text{and} \quad w_0 = \sigma_1 w_1$$

The character σ_1 which is scanned now by the head of TM has distance 1. (3)

In the case of MA now, after the initial rules (the setting up the string rules) have been applied and a "B" has appeared at the end of the string, it is impossible that those rules may become applicable again. Thus before the application of any rule corresponding to TM moves (not leading to final states, - block A-), the string of MA will present the form:

$$"q_0 w B" \quad \text{or (since from (1) } \rightarrow w = \sigma_0 w_0) \quad "q_0 \sigma_0 w_0 B".$$

But because there is nothing to the left of q_0 but $\dot{\epsilon}$, we have $|\dot{\epsilon}| = 0$ as in (1).

The MA rule equivalent to TM move (m1) is rule (r_1) easily to be determined by the consideration that from among all rules, the only

one applicable¹ in this case is the rule containing $q_0\sigma_0$ in its LHS.

$$q_0\sigma_0 \rightarrow \sigma_0, (q_0) \quad q_0, (\sigma_0) \quad (r_1)$$

After the application of rule (r_1) the string is modified to:

$$" \sigma_0, (q_0) \quad q_0, (\sigma_0) \quad w_0 \quad B "$$

and since from (2) we have that $w_0 = \sigma_1 w_1$, this string becomes:

$$" \sigma_0, (q_0) \quad q_0, (\sigma_0) \quad \sigma_1 \quad w_1 \quad B " \quad (4)$$

$$\text{Also} \quad | \sigma_0, (q_0) | = 1 \quad (5)$$

We conclude from (2), (3), (4), (5), as it is obvious, that after the first move, if A is true then B is true.

We accept now that after K moves, if A is true then B is true.

After K moves, TM will be in state

$$q_0, (\sigma_0, \sigma_1, \dots, \sigma_{K-1})$$

its head scanning the character σ_K , with distance r_K , from the beginning of the tape which will now have the form:

1. Something important should be noted here concerning the applicability of MA rules corresponding to TM moves.

By the construction of MA there is one unique MA rule for every TM move. In the LHS of all these MA rules there is a marker representing the state of TM, and the position of the head of TM at the time when TM is simulated by MA. This of course has been described elsewhere in this chapter. The interesting thing however is that at any time there is only one marker of this kind in the string. This means that only one rule is applicable at any time. This rule includes in its LHS the marker existing in the string and the same character, which is to the right of the marker in the string, being to the right of the marker (in the rule).

$$"w_K' \sigma_K w_K BBB \dots"$$

$$\text{where: } \sigma_K \in (\Gamma \cup \{\varepsilon\}), \quad w_K', w_{K-1}, w_K \in (\Gamma - \{B\})^* \quad (6)$$

$$\text{and } w_{K-1} = \sigma_K w_K$$

At the same time, by the assumption we have made (following the inductive method) MA having made K applications of its rule (corresponding to TM moves not leading to final states), will have effected a modification of the string so that its form now will be:

$$"w_K' q_0, (\sigma_0, \sigma_1, \dots, \sigma_{K-1}) \sigma_K w_K B" \text{ with } |w_K'| = r_K \quad (7)$$

We shall prove now that if statement A is true after K+1 TM moves B will also be true after K+1 applications of the MA rules where the corresponding TM moves do not lead to final states).

From (6) we see that the next move, K+1, for TM is:

$$\delta(q_0, (\sigma_0, \sigma_1, \dots, \sigma_{K+1}), \sigma_K) = (q_0, (\sigma_0, \dots, \sigma_K), \sigma_K, [q_0, (\sigma_0, \dots, \sigma_{K-1})], d) \quad ,d)$$

$$\text{where } d \in \{L, R\}. \quad (m_{K+1})$$

d = L : After the above move the tape will have the form

$$"w_{K+1}' \sigma_{K+1} \sigma_K, |q_0, (\sigma_0, \dots, \sigma_{K-1})| w_K BBB \dots" \quad (8)$$

$$\text{where } \sigma_{K+1} \in (\Gamma - \{B\})$$

$$w_{K+1}' \in (\Gamma - \{B\})^+$$

$$\text{and } w_{K+1}' \sigma_{K+1} = w_K' \text{ (this will happen if and only if } d=L).$$

We also substitute $\sigma_K, [q_0, (\sigma_0, \dots, \sigma_{K-1})] w_K$ by w_{K+1}

and then the tape will take the form:

$$"w_{K+1}' \sigma_{K+1} w_{K+1} BBB \dots"$$

with TM in state q (where $q = q_0(\sigma_0, \dots, \sigma_K)$) its head scanning the character σ_{K+1} .

The distance of the previously scanned symbol was r_K . But now the distance of the scanned square σ_{K+1} is r_{K-1} as (9) the head simply moved to the left just one square.

$$d = R$$

The tape will have the form:

$$" w'_K \sigma_K, [q_0, (\sigma_0, \sigma_1, \dots, \sigma_{K-1})] \sigma_{K+1} w_{K+1} BBB \dots "$$

$$\text{where } \sigma_{K+1} \in \Gamma, \quad w_{K+1} \in (\Gamma - \{B\})^* \quad (10a)$$

$$\text{and } w_K = \sigma_{K+1} w_{K+1}$$

This can happen only if $d = R$.¹

$$\text{We now substitute } w'_K \sigma_K, [q_0, (\sigma_0, \dots, \sigma_{K-1})] \text{ by } w'_{K+1} \quad (10b)$$

and the tape will assume the form:

$$" w'_{K+1} \sigma_{K+1} w_{K+1} BBB \dots " \quad (10c)$$

The machine is now scanning the character σ_{K+1} , being in state

$$q = q_0(\sigma_0, \sigma_1, \dots, \sigma_K) \quad (10d)$$

The head has moved one square to the right pointing now to the square with the character σ_{K+1} . The previous

1 We may note that if $d = L$ $w'_K = w'_{K+1} \sigma_{K+1}$
and if $d = R$ $w_K = \sigma_{K+1} w_{K+1}$

We do not use two symbols σ'_{K+1} and σ_{K+1} in the above cases, because in any case the next symbol to be scanned has to be σ_{K+1} by the symbolization we use. Of course this does not mean that σ_{K+1} in the case $d=L$, is the same as σ_{K+1} in the case $d=R$. Nevertheless, since the move is defined separately for the two distinct cases (with $d=L$ only or $d=R$ only) there can be no confusion as σ_{K+1} will appear only after the move.

distance was r_K so the new distance will be r_{K+1} (11)

If now for move m_{K+1} in TM we have $d = L$, then we will have for MA from (8) $\rightarrow w'_K = w'_{K+1} \sigma_{K+1}$ and the string will be:

$$" w'_{K+1} \sigma_{K+1} q_o, (\sigma_o, \sigma_1, \dots, \sigma_{K-1}) \sigma_K w_K B "$$

Thus we see that if there is a rule applicable at all, it must contain $q_o, (\sigma_o, \dots, \sigma_{K-1}) \sigma_K$ in its LHS. It follows that the unique rule must be:

$$\sigma_{K+1} q_o, (\sigma_o, \dots, \sigma_{K-1}) \sigma_K \rightarrow q_o, (\sigma_o, \dots, \sigma_{K-1}, \sigma_K) \sigma_{K+1} \sigma_K, [q_o, (\sigma_o, \dots, \sigma_{K-1})]^{B_K} (r_{K+1})$$

where $B_K \in \{B, \hat{e}\}$

The reason why we use B_K here, is that $\sigma_K \in \Gamma$ and $w_K \in (\Gamma - \{B\})^*$, so that if $w_K = \hat{e}$, σ_K may possibly be a blank symbol. Of course in this case there cannot be another B at the end of the string.

As a result of the application of rule r_{K+1} , the string has been modified to:

$$" w'_{K+1} q_o, (\sigma_o, \dots, \sigma_K) \sigma_{K+1} \sigma_K, [q_o, (\sigma_o, \dots, \sigma_{K-1})]^{w_K B_K} "$$

and since from (8) it follows that $\sigma_K, [q_o, (\sigma_o, \dots, \sigma_{K-1})]^{w_K} = w_{K+1}$

and $q = q_o, (\sigma_o, \dots, \sigma_K)$ the string will have the form,

$$" w'_{K+1} q \sigma_{K+1} w_{K+1} B " \quad (12)$$

because $|w'_K| = r_K$ and $|\sigma_{K+1}| = 1$

and $w'_K = w'_{K+1} \sigma_{K+1} \rightarrow |w'_{K+1}| = r_K - 1$ (13)

We conclude from (8), (9), (12), (13) (where $d = L$), that if A is true B is true.

If now in the TM move m_{K+1} $d = R$, we have:

$$w_K = \sigma_{K+1} w_{K+1}$$

Thus the string before the $K+1$ application of the MA rules will be:

$$" w_K' q_o, (\sigma_o, \dots, \sigma_{K-1})^{\sigma_K} w_K B "$$

Now if a rule is applicable at all, it must contain in its LHS

$q_o, (\sigma_o, \dots, \sigma_{K-1})^{\sigma_K}$. Such a rule of course does exist, and

corresponds to the m_{K+1} th TM move, where $d = R$. The rule is

$$q_o, (\sigma_o, \dots, \sigma_{K-1})^{\sigma_K} \rightarrow \sigma_K, [q_o, (\sigma_o, \dots, \sigma_{K-1})] q_o, (\sigma_o, \dots, \sigma_{K-1}, \sigma_K)^{B_K}$$

where again $B_K \in \{B, \dot{B}\}$. We repeat here that if $B_K = B$, there must not be another B (as $\sigma_K = B$ and $w_K = \dot{B}$) at the end of the string.

After the application of this rule, the string will have the form:

$$" w_K' \sigma_K [q_o, (\sigma_o, \dots, \sigma_{K-1})] q_o, (\sigma_o, \dots, \sigma_K)^{w_K} B "$$

In addition, since we have

$$w_{K+1}' = w_K' \sigma_K q_o, (\sigma_o, \dots, \sigma_{K-1})$$

$$w_K = \sigma_{K+1} w_{K+1}$$

$$q = q_o, (\sigma_o, \dots, \sigma_K)$$

from (10b), (10a), (10d) respectively, the form of the string will be:

$$w_{K+1}' q \sigma_{K+1} w_{K+1} B \quad (14)$$

$$\begin{aligned}
 \text{Because } |w'_{K+1}| &= |w'_K \sigma_K, [q_0, (\sigma_0, \dots, \sigma_{K-1})]| = \\
 &= |w'_K| + |\sigma_K, [q_0, (\sigma_0, \dots, \sigma_{K-1})]|,
 \end{aligned}$$

and since by our inductive hypothesis (7),

$$|w'_K| = r_K \quad \text{and}$$

$$|\sigma_K, [q_0, (\sigma_0, \dots, \sigma_{K-1})]| = 1$$

$$\text{we finally have: } |w'_{K+1}| = r_K + 1 \quad (15)$$

Thus from (10c), (10d), (14), (15) and from $d = R$, it follows that if for the (m_{K+1}) th move of TM, A is true then B is true. Since the same result has been proved for $d = L$, the inductive proof has been completed and whenever A is true B is true. The proof $B \rightarrow A$ is analogous.

6. ACCEPTANCE AND REJECTION OF A GIVEN STRING BY TM AND MA.

We can prove now that TM and MA are equivalent (that is they will behave in the same way if they are given the same input).

But before proving this, it should be clearly explained what we shall mean by saying "a word w is accepted, or rejected by a Turing Machine or a Markov Algorithm".

In the case of a Turing Machine, a word w is accepted by the machine, if and only if by placing w at the beginning of the tape when the machine is in the initial state q_0 and its head is scanning the first character of the input string (i.e. the one located at the leftmost square of the tape), the machine will eventually halt in a final state.

If the Turing Machine, under the same conditions halts in a non final state (and this happens for a pair $q_i \in (\bar{K}-F)$, $\gamma \in \Gamma$ if $\delta(q_i, \gamma)$ is undefined) the word w is rejected.

In the case in which the Turing Machine never halts (i.e. when it loops), w is again considered to be rejected.

As regards a Markov Algorithm now, it will be said to accept a word w , if after a certain number (possibly zero) of applications of its rules upon the word w , the algorithm terminates. On the other hand, if the algorithm does not terminate, we shall say that the string w is rejected. It may be noticed here that regarding a Markov Algorithm, there is no distinction corresponding to that between the two rejecting cases of a Turing Machine, the reason being that we cannot distinguish between terminating accepting rules and terminating non accepting rules of a Markov Algorithm. Thus always every kind of termination of a Markov Algorithm means acceptance of the input string w .

It is known (from its definition by Markov) that a Markov Algorithm terminates in one or the other of the following two ways:

- a: After the application of a terminating (concluding) rule,
- b: If the next rule tested is the last one, and is found to be inapplicable.

In both these two cases the algorithm is considered to have accepted the relevant input word.

So we can see that a Turing Machine can simulate a Markov Algorithm for the same input string w :

- a: by terminating in a final state and accepting, when the Markov Algorithm terminates,
- b: by looping and rejecting if the Markov Algorithm loops.

THEOREM 2.2

If a word $w \in \Sigma^*$ is accepted (or rejected) by TM, then w will be also accepted (or rejected respectively) by MA.

Proof

From the very construction of MA, we know that, given an input string $w \in \Sigma^*$, after a number of applications of the initial rules and before a rule belonging to blocks A or B has been applied, w will have been modified to " $q_0 wB$ ". Beginning from this point, MA will start the strict simulation of the moves of TM, by only applying rules which correspond to TM moves. This is secured by reason of the inapplicability of the initial rules after the application of the rule " $B' \rightarrow B$ ", as no other B' exists in the string. It is also true that after the application of an MA rule (corresponding to one of the TM moves) has taken place there will be another rule of this same kind applicable as long as there is a next move for TM. This again is of course safeguarded by the construction of MA itself.

A₁. If now, after a finite number of executions of its moves, TM halts, this can only happen because the input string w has been modified to $w' = w_1 \gamma w_2$ (where $w_1, w_2 \in (\Gamma - \{B\})^*$ and $\gamma \in (\Gamma - \{B\})$, and TM is found either

- a. In a final state $p_i \in F$, with its head pointing to a symbol $\gamma \in \Gamma$ (in one of the squares of its tape).
- or b. In a non final state $t_i \in (\bar{K} - F)$ scanning a symbol $\gamma \in \Gamma$, with $\delta(t_i, \gamma)$ undefined.

In both cases TM will halt, but accepting the string in case (a) and rejecting it in case (b) and its tape will be:

$$w_1 \gamma w_2 \text{ BBB } \dots$$

MA now, in its simulation of TM, will have modified the string " $q_0 wB$ " to

for case (a): $a' w_1 p \gamma w_2 B$ where $p \in \Gamma$ is the marker used $\forall p_i \in F$ of TM.

for case(b): $b' w_1 t_i \gamma w_2 B \quad t_i \in (\bar{K}-F)$

$$w_1, w_2 \in (\Gamma - \{B\})^*, \quad \gamma \in (\Gamma - \{B\})$$

In the case(a') the next rule applicable is the rule $P \rightarrow . \Lambda$ which is a terminating rule and so the algorithm terminates. This is necessarily so, since rules 0 and 1 are obviously inapplicable (no B' exists in the string), and none of the rules of blocks A and B can be applied either. The reason for this latter inapplicability is that all rules belonging to block A (these correspond to TM moves not leading to final states) have in their LHS a marker q where $q \in \bar{K}$. But at any time, there is only one marker of this kind in the string. Therefore, if there is another block A rule applicable, with " $uq\beta$ " as LHS (where $q \in \bar{K}$, $\beta \in \Gamma$ and $u \begin{cases} \in (\Gamma - \{B\}) & (1) \\ \in (\text{empty word}) & (2) \end{cases}$), it is necessary, according to the construction of MA) that a corresponding move

" $\delta(q, \beta) = (-, -, L)$ " for (1) or " $\delta(q, \beta) = (-, -, R)$ " for (2), should exist for TM.

But in such a case TM would have a next move to do after being in a final state, which is impossible being in contradiction to our hypothesis.

For the same reason the rules of block B (they correspond to moves of TM leading to final states) are inapplicable as well.

It follows that the next rule applicable is the terminating one $P \rightarrow . \Lambda$. After its application, MA terminates, and so accepts w as TM did for this case.

In case(b'), again, the rules 0 and 1 are inapplicable as before. None of the rules of blocks A and B are applicable either, because t_i is the only marker (of those belonging to \bar{K} , in fact $t_i \in (\bar{K}-F)$), in the string at this stage.

Therefore, if there was a next applicable rule from block A, the LHS of this rule must have been of the form " $ut_i\alpha$ " where $u \in (\Gamma - \{B\})$ or $u = \epsilon$ (empty word). Now in this case, there should again exist a corresponding TM move of the form $\delta(t_i, \alpha) = (-, -, {}^L_R)$, for which the MA rule was constructed. But from our hypothesis(b) $\delta(t_i, \alpha)$ is undefined, and hence there cannot be such an MA rule. Evidently, the rules of block B are also inapplicable for the same reason.

The rule " $P \rightarrow \Lambda$ " is inapplicable as well, because no P exists in the string, as none of the rules of block B were found applicable before.

Finally the rule " $B \rightarrow C$ " (which becomes applicable once rule 1 : " $B \rightarrow B$ " has been applied) is always applicable if a B exists in the string. Thus we can see that whenever rule " $P \rightarrow \Lambda$ " is next tested and found inapplicable, rule $B \rightarrow C$ is to be applied.

After this application the only change in the string is that "there is not a B", so the next rule applicable is rule " $C \rightarrow C$ ". From this point onwards this rule is repeatedly applied for ever, as it is always applicable and all the rules preceding it are inapplicable. Hence the next rule " $\Lambda \rightarrow q_0 B$ " (always applicable) is never tested, and in this case MA never terminates thus rejecting w similarly to TM (TM rejected the string).

We have added in the construction of MA, the rule " $C \rightarrow C$ ". We would have the same result by changing the previous rule to " $B \rightarrow B$ " and not using the rule " $C \rightarrow C$ " at all. But by using C, we are in the position to know after the first appearance of a C in the string that MA will never terminate and that the input string is indeed rejected. This of course has only (or mostly) practical significance, because theoretically the string is in any case rejected as MA does not terminate.

A₂. Finally, we will examine the case in which TM never halts.

In this case there will always be a next move for TM. If this happens, TM rejects the input string w .

From the previous cases, it is already known that MA cannot apply its initial rules "0" and "1" again once rule "1" has been applied. It is also established that for every TM move there is a corresponding MA rule in blocks A or B. Naturally we are not interested here in the rules of block B (the ones which correspond to TM moves leading to final states) because we are going to show that, in this case, the rules of block A only will be applied for ever (and hence rules belonging to block B will have never occasion to be tested regarding their applicability).

After an application of any one of the rules of block A, a marker $q \in (\bar{K}-F)$, and only one of this kind, will exist in the string of MA. Since every such q has on its right a character $\beta \in \Gamma$, (the same one which would be scanned by TM in state q , for this case) and since $\delta(q, \beta)$ is always defined (the reason being that TM has always a next move not leading to a final state, and similarly for MA), there will always be a next rule, in block A with " $\eta q \beta$ " in its LHS $(\eta \begin{cases} \in (\Gamma - \{B\}) \\ \cdot \\ \in (\text{empty string}) \end{cases})$.

Consequently MA, applying for ever rules of the block A will never terminate and the string w will be rejected just as in the case of TM.

THEOREM 2.3

If a word $w \in \Sigma^*$ is accepted (or rejected) by the constructed MA, it will be also accepted (or respectively rejected) by TM.

Similarly, to what was said above, in the case of MA the input string \dot{w} after the application of rules "0" and "1" will have been modified to " $q_0 w B$ ". From this point onwards only rules of block A

and possibly one from block B will be applied, as rules "0" and "1" become inapplicable.

Every rule from blocks A and B contains a marker $q \in (\bar{K}-F)$ in its LHS; in addition, every rule of block A (but not of B) contains a marker $q' \in (\bar{K}-F)$, and every rule of block B (but not of A) a marker $p \in (A-\bar{K})$, in their RHS (possibly $q = q'$). However, at any time, there is only one marker $p \in (KU\{P\})$ in the string. This marker p has to its right some character $\gamma \in \Gamma$. If a next rule is applicable from blocks A or B, the LHS of this rule must contain the string " $n\gamma$ " (where n is empty or $n \in (\Gamma-\{B\})$). If there is no such string among the LHS's of the rules incorporated in the blocks A and B, there is no next rule applicable to be found among these rules.

In this latter case the string will have been modified to either:

- a. $w_1 P \gamma w_2^B$ Here the presence of P indicates that one of the rules of block B has been applied.

or:

- b. $w_1 t_i \gamma w_2^B$

where $w_1, w_2 \in (\Gamma-\{B\})^*$, $\gamma \in (\Gamma-\{B\})$, $t_i \in (\bar{K}-F)$ and $P \in \Gamma$ a special character defined in the construction of MA.

In case (a) there are no other rules applicable before rule " $P \rightarrow \Lambda$ ". This rule is applied and MA terminates accepting w.

In case (b) there is no applicable rule preceding rule " $B \rightarrow C$ " which is consequently applied. The next applicable rule is rule " $C \rightarrow C$ ", which will be applied for ever as all the other rules preceding it are inapplicable. This means that the algorithm does not terminate; therefore it rejects w. We note however here the presence of a "C" in the string.

Let us now try to infer the behaviour of TM according to the way that MA has handled the input string.

MA has been so constructed that at any time there will be a

marker (and only one) $q \in \bar{K}$ in its string indicating the corresponding state of TM, given of course that the TM is working on the same initial string. This marker must have also always to its right a character $\gamma \in \Gamma$, which would be the same with the character correspondingly scanned by TM's head. Thus if there does exist for TM a next rule applicable from blocks A or B, its LHS would be of the form " $\eta q \gamma$ " where $\eta \in (\Gamma - \{B\})$ or $\eta = \epsilon$ as explained above. But by the MA construction, every one of the MA rules in blocks A and B corresponds to some one of the TM moves; in particular, the rule having as LHS the string " $\eta q \gamma$ " will correspond to the move: $\delta(q, \gamma) = (-, -, d)$ where $d \in \{L, R\}$. It is thus evident that whenever there is a next rule for MA, there will be a unique corresponding move for TM.

a'. In case (a) now the existence of "P" indicates that a rule of the block B has been applied, and since the rules of this block correspond to TM moves which lead to final states, we know that TM in this case will be in a final state, without a next move to follow. So TM will accept w just as MA did in case (a) after the application of its terminating rule.

b'. In case (b), MA did not terminate. So none of the rules of block B had been applied. (We know this, because, if one such rule had been applied, the special character P would have been present in the string, and MA would have terminated as in case (a).) Since the rules of block B are the only ones which correspond to TM moves leading to final states, we see that in no wise can TM be in a final state. So we can say that TM rejects the string w in this case, similarly to MA. It does not really matter whether TM halts or loops,

because both looping and halting in a non-final state are considered as rejections of the string by the machine.¹

- c. A third possibility for MA, is to loop, being continuously confined to the application of rules belonging to block A alone. This of course can be easily detected, since rules " $B \rightarrow C$ " and " $C \rightarrow C$ " will in this case remain permanently inapplicable, and hence the symbol C will never appear in the string.

In the present case, which anyway is covered by our previous comment, we know that since MA never terminates, it is considered to reject the input string w.

- c'. As for TM, since there is always a next rule for MA in block A, there will always be a next move corresponding to

1 If in examining the string of MA, we observe the occurrence of the symbol C after a finite number of applications of its rules, we infer that MA is not applying any more rules belonging to block A and hence, since the rules of block B are inapplicable in this case, MA must be applying rules which do not correspond to TM moves: in fact rule $C \rightarrow C$ is being applied for ever, since all the rules preceding it are inapplicable.

Now, since by the construction of MA, MA rules corresponding to TM moves must belong to the blocks A and B only; and, to put the point conversely, as there are no corresponding TM moves for all the other rules of MA; and finally, as the rules of block B never become applicable in our present case it follows that logically TM should halt. This must happen because otherwise, if TM would not halt, there would have been always available next moves corresponding to block A rules only - in which case TM would loop.

But MA should then have had in block A rules corresponding to these TM moves, and in such a case the rules " $B \rightarrow C$ " and " $C \rightarrow C$ " would have never become applicable, because one of the rules in block A would always be applicable.

This of course is in contradiction to our hypothesis that a C has in fact appeared in the MA string. Hence, since none of the MA rules contained in block A can be applicable any more, TM must also halt in this case, in a non final case.

The string w is then rejected by TM (as has been deduced above).

this MA rule (by construction). So TM will loop and will never terminate. In this case, too, then, TM rejects w .

7. A LABELLED MARKOV ALGORITHM SIMULATING TM - (LMA).

For any given TM an equivalent Labelled Markov Algorithm LMA, could be constructed.

An efficient LMA will be a Markov Algorithm with labels, simulating TM in exactly the same way as MA, but with less searching for discovering the next applicable rule in each case.

In TM "the next move" is always known immediately without any search through the TM moves. This happens because the position in the tape where a change is going to take place is always pointed at by the machine's head. Of course in the case of MA the same position is specified at any time by the unique marker $q \in \bar{K}$ in the string, which also indicates the corresponding state of TM. Nevertheless, in the case of MA, it is not known where in the string this marker is or indeed which of all the $q_i \in \bar{K}$ markers is the "wanted" one (i.e. the existing one in the string) after every application of a rule. Besides, from among all rules containing q_i in their LHS, one and only one is applicable at a time, and this is the one which contains " $\dots q_i \gamma$ " in its LHS, where $\gamma \in \Gamma$ is the character in the string which is the same with the one to be found on the right of marker q_i .

It is thus evident that in order to determine the next rule applicable in the case of MA, it is necessary to carry out some search procedure. The actual amount of search can however be reduced considerably (especially in the case in which the number of states of TM is very large) by the construction of an LMA.

The LMA which would have to be constructed must have the same rules as MA, the difference being that it will of course be labelled. The most important MA moves are those contained in blocks A, B (as these are the ones which strictly correspond to TM moves). So we shall start from them.

The moves of TM can be ordered according to the subscripts i for all states $q_i \in (\bar{K}-F)$. Thus regarding the moves (a) and (b) of TM:

$$\begin{aligned} \text{a: } \delta(q_i, \gamma) &= (q_i', \gamma', \begin{smallmatrix} L \\ R \end{smallmatrix}) & \text{where } q_i, q_j \in (\bar{K}-F), \\ & & q_i, q_j \in \bar{K}, \\ \text{b: } \delta(q_j, \gamma) &= (q_j', \gamma', \begin{smallmatrix} L \\ R \end{smallmatrix}) & \gamma \in \Gamma, \gamma' \in (\Gamma - \{B\}). \end{aligned}$$

if $i < j$, move (a) is written first

if $i > j$, move (b) is written first

if $i = j$: then

if $i' < j'$ move (a) is written first

if $i' > j'$ move (b) is written first

if $i' = j'$ the order is immaterial.

The same procedure can be followed in ordering the MA rules corresponding to the TM moves. These rules belong to blocks A and B.

Thus regarding the rules (a') and (b') of MA,

$$\begin{aligned} \text{a': } uq_i\gamma &\rightarrow "...q_{i'}..."^1 & \text{where } \gamma \in \Gamma \\ & & u \in \{\Gamma, \epsilon\} \quad \epsilon = \text{empty string} \\ \text{b': } uq_j\gamma &\rightarrow "...q_{j'}..." \end{aligned}$$

and the same ordering is to be effected, depending on the particular values of i, j, i', j' , as in the case of the TM moves.

We know now, that after a move has been executed by TM, eg.

$$\delta(q_i, \gamma) = (p_i, \cdot),$$

the next move (if such an one is defined, and if $p_i \in (\bar{K}-F)$) will belong to a certain sub-block of moves, normal to the one for which the state q in every $\delta(q, \dots)$ has the value p_i .

The same ordering can be achieved as in the case of MA, for rules in blocks A and B. E.g. after the rule

$$uq_i\gamma \rightarrow \dots p_i \dots \quad (\text{where } u \in [(\Gamma - \{B\}) \cup \{\epsilon\}])$$

has been applied, the next rule tested will be the first rule from the sub-block of rules containing the marker p_i in their LHS.

This is of course to be achieved by simply attaching a label to the end of the rule of our example. This label will be written at the end of the rule and will function as a pointer pointing to

1 The TM move

$$\begin{aligned} \delta(q, \gamma) &= (q', \gamma', d) & \text{where } q, q' \in \bar{K} \\ & & \gamma \in \Gamma \\ & & \gamma \in \{\gamma_1, \gamma_2\} / \gamma_1, \gamma_2 \in (\Gamma - \{B\}) \end{aligned}$$

has as corresponding MA rule:

$$uq\gamma \rightarrow u_1\gamma_1 q' B_1 u_2\gamma_2 B_2$$

where $u \in \{\Gamma - \{B\}, \epsilon\}$ (ϵ = empty string).

$$\text{iff } u_1 = \epsilon, \gamma_1 = \epsilon \rightarrow u_2, \gamma_2 \in \Gamma$$

$$\text{iff } u_2 = \epsilon, \gamma_2 = \epsilon \rightarrow u_1, \gamma_1 \in \Gamma$$

$$\text{iff } \gamma = B \rightarrow (\text{iff } B_1 = \epsilon \rightarrow B_2 = B)$$

$$\text{iff } \gamma \neq B \rightarrow (\text{iff } B_2 = B \rightarrow B_1 = \epsilon)$$

(iff means "if and only if").

Furthermore, in cases that we are only interested in the existence of a marker q' in the RHS, we can discard the above formulation of the RHS of the rule, preferring the following one:

$$uq\gamma \rightarrow \dots q' \dots$$

the next rule tested. Thus, if K is the K th rule written in accordance with the ordering we have instituted (where $0 \leq 1 < n$, n being the number of MA rules) then, for any rule having the label K at its end it is assumed that, once it has become applicable, the next rule to be tested will be rule K .

In this way, the moves of TM not leading to final states and the corresponding MA rules belonging to block A will take the form:

$$\begin{aligned}\delta(q_i, \alpha) &= (p_i, b, R) \rightarrow q_i \alpha \rightarrow b p_i, K_{p_i} \\ \delta(q_j, \alpha) &= (p_j, b, L) \rightarrow \sigma q_j \alpha \rightarrow p_j \sigma b, K_{p_j} \\ \delta(q_i, B) &= (p_i, \gamma, L) \rightarrow q_i B \rightarrow \gamma p_i, B, K_{p_i} \\ \delta(q_j, B) &= (p_j, \gamma, L) \rightarrow \sigma q_j B \rightarrow q_j \sigma \gamma B, K_{p_j}\end{aligned}$$

where K_{p_i} , K_{p_j} , $K_{p_i'}$, $K_{p_j'}$ are the labels of the next rules tested,

$$\begin{aligned}q_i, q_j, q_i', q_j', p_i, p_j, p_i', p_j' \in \bar{(K-F)} \\ \alpha \in \Gamma, b, \gamma \in (\Gamma - \{B\})\end{aligned}$$

All the rules of block B (which correspond to TM moves leading to final states) will naturally be assumed to point to the rule " $p \rightarrow \Lambda$ " as the next one to be tested.

We can now construct an LMA equivalent to TM, and, of course, to MA. This can be done since what the difference between MA and LMA amounts to really, is that MA has to test for applicability some rules, which are known to be inapplicable in LMA, and are accordingly not tested by it.

LMA

0 : $\Lambda \rightarrow q_0 B', 1;$

1 : $B' \xi \rightarrow \xi B', 1;$

2 : $B' \rightarrow B;$

- . rules of
- . blocks A
- . and B (to-
- . gether or-
- . dered)

$n+K+2$

$n+K+3$: $P \rightarrow . \Lambda ;$ (no label)

$n+K+4$: $C \rightarrow C, n+K+4;$

8. CONCLUSION

We have now completed the proof of the proposition that for any given TM, an equivalent MA (and, indeed, an equivalent labelled MA) can be constructed, i.e. one behaving in an exactly analogous way and leading to exactly the same results as the TM.

It has also been shown in the first chapter of this dissertation, that the converse of the above stated proposition holds equally good, namely that for any given MA an equivalent TM can be constructed exactly simulating the algorithm's behaviour.

We therefore conclude now that the equivalence of Turing Machines and Markov Algorithms has been thereby demonstrated and thus the one of these two devices can do exactly what the other can do and nothing more.

CHAPTER III .

NONDETERMINISM IN MARKOV ALGORITHMS AND TURING MACHINES

1. INTRODUCTION

In what follows we will study nondeterminism in Turing Machines and Markov Algorithms. Since the results of the inquiries concerning Nondeterministic Turing Machines are well known, we shall only briefly describe what a Nondeterministic Turing Machine is and simply state that for every Nondeterministic Turing Machine there is an equivalent deterministic one and vice versa. On the other hand, we shall discuss in more detail Nondeterministic Markov Algorithms, because, as far as we know, the relevant literature on the subject is rather meagre.

Thus, we will firstly describe and define what a Nondeterministic Markov Algorithm is, giving examples for different kinds of Nondeterminism in Markov Algorithms; and then we will proceed to show, using the equivalence of a Nondeterministic Turing Machine to a Deterministic one, that an unrestricted Nondeterministic Markov Algorithm is equivalent to a Deterministic one. This proposition will be proved by the construction of a Nondeterministic Turing Machine for a Nondeterministic Markov Algorithm. Finally, we will also construct an equivalent Nondeterministic Turing Machine for an unrestricted Nondeterministic Markov Algorithm. This construction will be conducted in a very similar and quite analogous way to the corresponding constructions for Deterministic models in the previous chapters.

2. NONDETERMINISTIC GRAMMATICAL DEVICES.

As a nondeterministic Grammatical Device we define a Grammatical Device which may choose its every next action from a finite set (possibly empty) of different actions defined in each case for a given internal instruction (e.g. state) and/or external information

(e.g. character read).

For example the next action of a finite state machine is its next state and its head move to the right by one square (on the tape); this second component action however is obviously independent of the information (as the head always moves right, but with the exception that it halts after the whole input string has been scanned).

The next action of a Nondeterministic Grammatical Device (NGD), is chosen arbitrarily from a finite and possibly empty set of next actions, (defined for the same input information), and no priority is given to any one action in this set against the others.

When a string is given as input to a Nondeterministic accepting Grammatical Device, it will be accepted if and only if there is at least one path of next action choices leading to the satisfaction of an acceptance criterion, included in the function definition of the Nondeterministic accepting Grammatical Device in question.

3. A NONDETERMINISTIC TURING MACHINE

A Nondeterministic Turing Machine (NTM), is defined as follows:

$$NTM = (\bar{K}_N, \Sigma, \Gamma, \delta_N, q_0, F_N) \quad \text{where:}$$

\bar{K}_N is the finite set of the NTM's states

Σ is a finite set of input symbols (or characters)

Γ is the finite set of all symbols used

F_N is the set of final states, $F_N \subseteq \bar{K}_N$

q_0 is the initial state, and

δ_N a mapping from $(\bar{K}_N \times \Gamma)$ to subsets of $(\bar{K}_N \times (\Gamma - \{B\}) \times \{L, R\})$.

Thus we see from the very definition of the function δ_N that for

the same state-character pair a NTM may have a number of options for its next action, contrary to what happens in a Deterministic TM, where the machine has a unique next action for every state-character pair.

In addition to the above definition of a NTM, some authors require that a NTM should have a two way infinite tape. This however does not materially alter the above definition because, as can be shown, every Turing Machine with a two way infinite tape can be converted to a NTM with a one way infinite tape.¹

It should be noted that, as for other classes of automata too, for every NTM an equivalent Deterministic TM can be constructed and vice versa. In other words it is known that NTM's and Deterministic TM's are equivalent.²

4. NONDETERMINISTIC MARKOV ALGORITHMS

We have already defined a Deterministic Markov Algorithm as a system:

$$DMA = (A, \Sigma, P, N_p).$$

The question is: What should we add or alter in this definition, in order to define a Nondeterministic Markov Algorithm (NMA)?

Similarly to what happens in all Nondeterministic Grammatical Devices, every next action of NMA must be chosen arbitrarily from a set of (possibly empty) next actions. As has already been explained, the possible actions of a Deterministic Markov Algorithm are:

- 1 Hopcroft and Ullman "Formal Languages and their relation to Automata".
- 2 Hopcroft and Ullman "Formal Languages and their Relation to Automata". Zohar Manna: Mathematical Theory of Computation.

1. Which is (if there is any) the uniquely determined single rule to be tested next or is it the case that DMA does terminate now?
2. The applicability search for the rule being tested.
3. The application of an applicable rule.

Markov in his definition of Normal Algorithms recognises that the LHS of a rule may be included in a given string more than once.

"... Therefore", he says, "it has to be indicated for which of its entries the right-hand member of the formula is to be substituted... We shall agree ... that we shall apply the corresponding substitutions to the first entry of the left-hand member of the formula in the word ..."

If in our definition of a NMA we do not include the set of instructions N_p , then this will imply that the next rule to be tested in each case is to be chosen arbitrarily. On the other hand, if we exclude Markov's assumption that the applicability search for a rule starts at the beginning (left most character) of the string and if we allow it to start at any position, chosen arbitrarily, then the LHS of an applicable rule may be substituted not in the place of its first occurrence in the string, by the LHS of this rule, but in the place of any one of its occurrences.

Thus, taking this into account, we can define now as a Genuine Nondeterministic Markov Algorithm, (GNMA), a Nondeterministic Rewriting System GNMA, where:

$$\text{GNMA} = (A, \Sigma, P).$$

The finite sets A, Σ, P are the same as the ones used in the definition of a Deterministic MA.

To complete the definition we should add that, in each case, an applicable rule can be applied on the given string by substitution

of an occurrence of its LHS by its RHS - there being no preference for any particular occurrence of the LHS (if there are more than one) in the string. We renamed the defined NMA as a Genuine NMA since we did not institute any restriction either on the way in which the "applicability search" is carried out or in the process for determining "the next rule tested".

By establishing various combinations of restrictions, other Nondeterministic MA models can be defined which will, therefore, not be Genuine Ones according to our definition.

However, none of these models, not even a Genuine NMA, is more powerful than a Deterministic MA as will be informally shown below.

4.1 Cases of "more or less Nondeterministic" Markov Algorithms

Having defined a Genuine NMA and a Deterministic Markov Algorithm, we can now distinguish other Nondeterministic MA models (not Genuine ones), which can be accordingly classified as lying between a Deterministic MA and a Genuine Nondeterministic MA. These Nondeterministic MA's are "more or less nondeterministic" (if this expression may be allowed) according to their degree of freedom against a Deterministic MA, or according to the serenity of their restrictions against the absolute freedom of a Genuine Nondeterministic MA.

Thus in the case of the following Nondeterministic MA models we assume that they can be constructed either by elimination or attenuation of a number of conditions involved in the definition of a Deterministic MA, or, on the other hand, by introducing restrictions in a Genuine Nondeterministic MA.

A. The next rule tested is not determined.

There are several models of Nondeterministic MA, for which the next rule to be tested each time is not determined.

In the "more nondeterministic" models of this type, every rule has exactly the same probability to be chosen as the next (in each case). But in the "less nondeterministic" models, of the same type, the next rule to be tested each time can be chosen from various subsets of the set of rules of the Algorithm.

An example of this type, is a Labelled Nondeterministic MA with rules of the form:

$$\text{LHS}_i \rightarrow \text{RHS}_i, \{i_1, i_2, \dots, i_{m_i}\};$$

Where, if n is the number of rules of the Algorithm for i, m_i, j, i 's, it is:

$$0 \leq i < n, \quad 0 \leq m_i < n \quad \forall i, \quad 1 \leq j \leq m_i \quad \text{and} \quad 0 \leq i_j < n \quad \forall j.$$

This means that whenever a rule i has been applied, the next rule tested will be chosen from a subset containing m_i ($0 \leq i < n$) of the rules of the Algorithm. All the rules in the subset $\{i_1, i_2, \dots, i_{m_i}\}$ have equal probability to be chosen as the ones next to be tested. As can be easily seen a number of different Labelled Nondeterministic MA can be constructed, (some "more" and some "less" nondeterministic) by giving different values to m_i ($0 \leq i < n$).

Thus if:

$$\text{a: } m_i = 0 \text{ or } m_i = 1, \forall i \quad 0 \leq i < n$$

we have a Labelled Deterministic MA.

$$\text{b: } m_i = n \quad \forall i : 0 \leq i < n$$

we have a Labelled Nondeterministic MA which is the "most nondeterministic" of this type.

- B. The applicability search, for any rule, does not start at a determined position of the string.

C. Rules with identical LHS's may have different RHS's

This type of Algorithms however, cannot be considered as Nondeterministic, if they are not also of type A or B.

In fact if an Algorithm is both of type A and of C, then it is "more nondeterministic" than its counterpart of type A which is not also of type C.

As we already said, the "most Nondeterministic Markov Algorithm" is a Genuine NMA. By attaching labels to a Genuine NMA, we have a Labelled Genuine NMA, if every of its rules apart from the terminating one is of the form:

$$\text{LHS}_i \rightarrow \text{RHS}_i, \quad \{j/j \text{ is a rule: } 0 \leq j < n\}; \quad (0 \leq i < n).$$

This can be written more explicitly:

$$\text{LHS}_i \rightarrow \text{RHS}_i, \quad 0, 1, 2, \dots, n-1; \quad (0 \leq i < n).$$

5. A NONDETERMINISTIC TURING MACHINE FOR A GNMA

Let us suppose that we have a Genuine Nondeterministic Markov Algorithm, "GNMA", defined as:

$$\text{GNMA} = (\Sigma, A, P)$$

where it is the case that the sets Σ, A, P are the same with those involved in the definition of the MA for which the equivalent Turing Machine TM was constructed (in the previous chapter).

(It should be remembered that the Alphabet $A = \{a_1, a_2, \dots, a_K\}$ has K elements and that the number of MA rules is n . We adopt the same convention here, for GNMA.)

Now for this given GNMA, we will construct a Nondeterministic Turing Machine, "NTM", in a way similar to the one we employed in constructing the TM equivalent to a given MA in the previous chapter.

Thus we define NTM as a system:

$$\text{NTM} = (\bar{K}_N, \Gamma, \Sigma, \delta_N, F)$$

where: Γ is the same finite set as in the case of TM.

F is the same one element set, as in the case of TM, comprising the unique NTM's final state.

$$\bar{K}_N = \bar{K} \cup \{Q(i, -1) / \forall i : 0 \leq i < n\},$$

where: \bar{K} is the set of states of TM and n the number of rules of MA and GNMA.

$$\delta_N = (\delta - \delta_u) \cup \delta_a$$

where: δ is the mapping function, that is the set of moves of TM.

δ_u is a subset of δ not included in δ_N .

δ_a is a subset of δ_N not included in δ .

5.1 Definition of sets δ_u and δ_a

The following moves, unwanted for NTM, are the only elements of

δ_u :

$$1. \quad \delta\{Q(i, w_i), \Omega_i\} = \{Q(\theta_i, 0), \Omega_i, R\} \quad , \quad \forall i : 0 \leq i < n$$

where $\Omega = T_i$ if $|LHS_i| \leq |RHS_i|$, and $\Omega = Y$ otherwise.

(If however we want to construct a Labelled GNMA, we do not exclude the moves of type 1 but we modify them to:

$$\delta\{Q(i, w_i), \Omega_i\} = \{Q([0, 1, 2, \dots, n-1], 0), \Omega_i, R\}.$$

$$2. \quad \delta\{Q(i, w_i+1), T_i\} = \{Q(i+1, 0), T_{i+1}, R\} \quad \forall i : 0 \leq i < n-1$$

$$3. \quad \delta\{Q(n-1, w_{n-1}+1), T_{n-1}\} = \{Q(n+2, 0), S, R\}$$

The number of moves of type 1 which immediately follow upon the application of each rule i , is n ; ($\forall i: 0 \leq i < n-1$).

The number of moves of type 2 which immediately follow upon the applicability search for an inapplicable rule i is $n-1$.

The move of type three is a single move.

The following moves included in δ_N , in addition to $(\delta - \delta_u)$, are the only elements of δ_a :

$$\begin{aligned}
 & \forall i : 0 \leq i < n \\
 \text{N1.a} : & \delta\{Q(i, w_i), \Omega_i\} = \{Q(\theta, -1), T_\theta, R\} \quad \forall \theta : 0 \leq \theta < n \\
 & \Omega_i = T_i \quad \text{if } |LHS_i| \leq |RHS_i| \quad \text{and } \Omega_i = "Y" \text{ otherwise.} \\
 & \forall i : 0 \leq i < n \\
 \text{N1.b} : & \delta\{Q(i, w_i), \Omega_i\} = \{Q(\theta, 0), T_\theta, R\} \quad \forall \theta : 0 \leq \theta < n \\
 & \text{and with } \Omega_i \text{ with same as above.}
 \end{aligned}$$

$$\begin{aligned}
 & \forall i : 0 \leq i < n \\
 \text{N2.3.a:} & \delta\{Q(i, w+1), T_i\} = \{Q(\theta, -1), T_\theta, R\} \quad \forall \theta : 0 \leq \theta < n \\
 & \forall i : 0 \leq i < n \\
 \text{N2.3.b:} & \delta\{Q(i, w+1), T_i\} = \{Q(\theta, 0), T_\theta, R\} \quad \forall \theta : 0 \leq \theta < n
 \end{aligned}$$

In all the above moves θ is the next rule tested and since it does not depend on the previously applied or tested rule (as in moves of type 1,2,3), it is to be chosen arbitrarily.

The number of moves of each of the above sets is $n \times n$, that is n^2 , as can be easily seen.

As can also be seen, if the moves of the sets N1.a and N2.3.a are the ones chosen by NTM, then the applicability search for rule i will not start from the square with the first character of the string, whereas if the moves of the sets N1.b and N2.3.b are those chosen, the applicability search for rule i will start with the first character of the string.

The following moves are also included in δ_a :

$$N4.a: \quad \delta\{Q(i,-1),\xi\} = \{Q(i,-1),\xi,R\}$$

$$N4.b: \quad \delta\{Q(i,-1),\xi\} = \{Q(i,0),\xi,R\}$$

$$N4.c: \quad \delta\{Q(i,-1),B\} = \{Q(i,w+1),B,L\}$$

where $\xi \in A$, $(A = \{a_1, a_2, \dots, a_K\})$ and $0 \leq i < n$.

The number of moves for sets N4.a and N4.b is $n \times K$, and for set N4.c is n .

The states $Q(i,-1)$, $(0 \leq i < n)$, are right moving states and they can bring the head of TM in any arbitrary position between the square with a T_i and the first blank square (included). Whenever NTM changes its state to $Q(i,0)$, (before a blank square has been scanned by its head), with its head scanning a certain square of the tape (lying between the square with T_i and the first blank square), the applicability search for rule i will start from this square of the tape. In the case in which TM remains in state $Q(i,-1)$ until its head scans the first blank square, it will change state to $Q(i,w+1)$, (a state which actually means inapplicability of rule i) and will drive its head to the square with the T_i .

Adding to the above that NTM will terminate only if it is in its final state, we complete the definition of NTM.

5.2 Equivalence of NTM and GNMA

In the first chapter a Deterministic Turing Machine TM was constructed so as to simulate a Deterministic Markov Algorithm MA. The proof of equivalence between GNMA and NTM is very similar to the proof (given in the first chapter) of equivalence between MA and TM. Thus we can state now that "For every Genuine Non-

deterministic Markov Algorithm there is a Nondeterministic Turing Machine capable of doing exactly what the Genuine Nondeterministic Markov Algorithm can do and no more".

6. A NONDETERMINISTIC MARKOV ALGORITHM FOR A NONDETERMINISTIC TURING MACHINE.

We have already defined a Nondeterministic Turing Machine.

For a Nondeterministic Turing Machine $NTM = (\Sigma_N, \bar{K}, \Gamma, \delta_N, q_0, F_N)$ we will now construct a NMA simulating NTM in a manner very similar and analogous to the one employed in the previous chapter concerning their deterministic counterparts.

There will be however some small but basic differences in the corresponding construction. We rewrite below the algorithm MA of the previous chapter, which was constructed so as to simulate TM.

0	:	$B'\xi \rightarrow \xi B'$	
1	:	$B' \rightarrow B$	
2	:		
.	:	block A	n rules corresponding to n TM moves not leading
.	:		to final states
n+1	:		
n+2	:		
.	:	block B	K rules corresponding to K TM moves leading to
.	:		final states
n+K+1	:		
n+K+2	:	$P \rightarrow .A$	} the unique terminating rule of MA
n+K+3	:	$B \rightarrow C$	whenever this rule will be applied, the next be-
n+K+4	:	$C \rightarrow C$	comes applicable and MA will never terminate,
			applying this rule ($C \rightarrow C$) for ever).
n+K+5	:	$A \rightarrow q_0 B'$	initial rule (for setting up the string), the
			first applicable rule of MA.

6.1 Differences in the corresponding constructions

The Deterministic MA was constructed in such a way that it was capable of doing exactly what TM could do and no more. This was achieved in the following way:

- a: For every move of TM, a corresponding rule was incorporated among the MA rules. This rule became applicable whenever, in the case of TM, its corresponding move became executable. But exactly the same can be done in the case of the construction of a Nondeterministic Markov Algorithm simulating a Nondeterministic Turing Machine. That is, we will construct (and include among the NMA rules) a NMA rule for every NTM move. Thus, whenever a move (or possibly moves) will become executable for NTM, a corresponding rule (or several rules, the same number as the TM corresponding moves) will simultaneously become applicable for MA. (It is assumed here that NTM and NMA are using the same path of next actions.)
- b: In the case of the deterministic models, after the application of an applicable rule or after the completion of the applicability search for an inapplicable rule, the next rule tested is uniquely defined. Thus the rule $n+K+5$ ($A \rightarrow q_0 B'$), is applied once only and cannot be tested again in the future. Similarly after the application of rule 1 ($B' \rightarrow B$), the rules 0 ($B' \xi \rightarrow \xi B'$) and 1 can never become applicable in the future. This means that after the unique application of rule 1, applicable rules can only be found among the ones contained in blocks A and B, or among the rules $n+K+2$, $n+K+3$, $n+K+4$. In that construction (deterministic MA for TM), the rules $n+K+5$, 0 and 1 do not add anything to the power of MA, but they merely give MA the ability to set up a given string itself. This modified input string to the main part of MA

can be regarded as a kind of counterpart of TM's head and tape (or indeed a part of TM's tape actually). The effect of this operation is that if the initially given string is "w", after the application of the setting up rules the string is modified to " $q_0 wB$ ".

The same effect could be achieved by the construction of two different Algorithms. The former which would only set up the string, is defined by:

$$0 : B'\xi \rightarrow \xi B'$$

$$1 : B' \rightarrow .B$$

$$3 : \Lambda \rightarrow q_0 B'$$

It is easy to see that this Algorithm if applied to a string $w \in \Sigma$, will modify it to " $q_0 wB$ " and terminate thereat.

The second Algorithm will be constructed from the rules of blocks A and B and from the rules $n+K+2$, $n+K+3$, $n+K+4$. The input string to this Algorithm will be the string " $q_0 wB$ ", if string w is the one initially given.

Alternatively, the second Algorithm alone can be used, provided that every string "w" is modified to " $q_0 wB$ " before being given an input to it.

Coming now to the construction of a Genuine Nondeterministic Markov Algorithm (GNMA), equivalent to a Nondeterministic Turing Machine (NTM), it should be noted that if we include the rules which set up the string, then GNMA will not be an exact NTM's simulator.

For the rule $n+K+5$ ($n \rightarrow q_0 B$), being always applicable, will be applied whenever it is tested and at any position of the string. This would make possible the appearance, in the string, of some "B" characters which may be followed by "non B" characters and the occurrence of more than one " q_0 ". By successive applications of

rules corresponding to NTM moves some of the q_0 's may be modified to q_i 's ($i \leq 0 < n$, where n is the number of TM's states). But corresponding to this state of affairs is the impossible situation of NTM being in several states simultaneously and having several heads scanning different squares of the tape.

In order to overcome this problem, we exclude the rules necessary for the setting up of the string and before we give to GNMA a string "w" as input, we modify it to " $q_0 wB$ ".

Thus GNMA will be constituted by the rules of blocks A and B and, in addition by the terminating rule ($P \rightarrow .A$). The rules ' $B \rightarrow C$ ', and ' $C \rightarrow C$ ' are not to be included, and consequently, a B will exist at the end of every output string whenever GNMA terminates. The difference from the deterministic MA model, (constructed in the previous chapter) as regards the rules of blocks A and B is that in the case of GNMA there may exist rules in blocks A or B or in both with identical LHS's and different RHS's. This will happen because some "state-character" pairs in NTM may result to more than one combinations of "next state-character printed - direction of head's move". For any of these combinations we should include a corresponding rule for GNMA.

GNMA will be Genuine, since we have introduced no restriction in it. The fact that the LHS of any of its rules can occur once only in the string, at any one time, does not constitute a proper restriction, because there is only one q_i ($0 \leq i < n$) in the string at a time and there is a q_i in the LHS of every one of the GNMA rules, except for the terminating one, whose LHS will make a unique appearance somewhere in the string, precisely when this rule becomes applicable.

On the other hand if, alternatively, we wish to keep the automation, so that the constructed Markov Algorithm should set up the

string itself, we cannot then have a GNMA, but only a restricted type of a Labelled NMA. In such a case the constructed Algorithm will be constituted as follows:

0	:	$\Lambda \rightarrow q_0 B', 1;$	
1	:	$B' \xi \rightarrow \xi B', 1;$	
2	:	$B' \rightarrow B, \{i / \forall i : 3 \leq i < n+K+3\}$	
3	:		>>
.	:		.
.	:	Block A	.
.	:		.
n+1	:		>>
n+2	:		>>
.	:		.
.	:	Block B	.
.	:		.
n+K+1	:		>>
n+K+2	:	$P \rightarrow \Lambda ;$	

We see that for every rule, beginning with rule 2 and going up to rule n+K+1, there are n+K labels (all the same) for the next rules to be tested. Thus, after the application of rule 2, the Algorithm can be regarded as a Labelled GNMA.

6.2 The constructed GNMA

We can now display the GNMA constructed as equivalent to NTM. It is not significant which rules should be written first and which next since every rule has the same possibility of being chosen as the next to be tested. We may therefore include first the rules which correspond to NTM moves not leading to final states, then the rules which correspond to NTM moves leading to final states and finally the terminating rule:

GNMA rules

0 :
 . rules corresponding to NTM moves not
 . block A leading to final states
 .
 n-1 :
 n
 . rules corresponding to NTM moves leading
 . block B to final states.
 .
 n+K-1 :
 n+K : $P \rightarrow . A$ terminating rule

Where n and K are the number of NTM moves respectively, not leading and leading to final states with each different combination of "next state-printed character - head's direction" for the same "state-character read" pair being considered as a separate move.

6.3 Equivalence of GNMA and NTM

The proof is very similar to the one followed in the previous chapter for the deterministic counterparts of GNMA and NTM. Thus we can now state that: "For every Nondeterministic Turing Machine there is a Genuine Nondeterministic Markov Algorithm capable of doing exactly what the Nondeterministic Turing Machine can do, and nothing more".

7. EQUIVALENCE BETWEEN NONDETERMINISTIC AND DETERMINISTIC MARKOV ALGORITHMS

We can now show what we set out to prove right at the beginning of this chapter, i.e. the equivalence between a GNMA and a DMA.

We know that:

- a: For every GNMA a NTM can be constructed capable of doing exactly what the GNMA can do and nothing more.

- b: For every NTM a DTM can be constructed capable of doing exactly what the DTM can do and nothing more.
- c: For every DTM a DMA can be constructed capable of doing exactly what the GNMA can do and nothing more.

From a,b,c it follows:

For every GNMA a DMA can be constructed capable of doing exactly what the GNMA can do and nothing more.

And since the construction of a GNMA for every DMA can be considered as trivial, we can finally say that:

"Genuine Nondeterministic Markov Algorithms and Deterministic Markov Algorithms are equivalent Grammatical Devices".

Evidently, all the restricted types of Nondeterministic Markov Algorithms cannot be more powerful than Genuine Nondeterministic Markov Algorithms; they also cannot be less powerful than Deterministic Markov Algorithms. And since DMA's and GNMA's are equivalent, this means that all the restricted types of NMA's must be equivalent to DMA's and GNMA's.

CHAPTER IV

CONSEQUENT RESULTS OF THE EQUIVALENCE

1. INTRODUCTION

In this chapter we will utilize results already obtained in the two previous chapters, in order to examine certain relationships holding between Markov Algorithms, Turing Machines and Types of Grammars. Although we will examine these Grammatical Devices from the recognizing point of view, the same technique could be followed if the Grammatical Devices were Generating ones.

More explicitly we will examine the relations between several types of Grammars and types of Turing Machines, as well as the relations between Grammars of the same type.

It can be easily seen, by their definitions, that the several types of Grammars can be regarded as restricted types of Markov Algorithms. Actually the Deterministic Grammars are restricted types of Deterministic Markov Algorithms and the Nondeterministic Grammars are restricted types of Nondeterministic Markov Algorithms.

2. EQUIVALENCE OF LABELLED AND UNLABELLED MARKOV ALGORITHMS..

As we have already proved, an equivalent Turing Machine can be constructed for every Labelled or Unlabelled Markov Algorithm (LMA or UMA).

In chapter I we constructed a Turing Machine, TM, to simulate a Markov Algorithm MA, and we did not specify whether MA was labelled or not.

The constructed TM, while simulating MA, with its corresponding set of moves after every application of any of the MA rules and before its first move of the applicability search for the next rule tested, executes the move:

$$\delta\{Q(i, w_i), \Omega_i\} = \{Q(\theta_i, 0), \Omega_i, R\}, \quad \forall i : 0 \leq i < n,$$

where if MA is labelled θ_i is dependent on the applied rule i and is the next rule tested, whereas if MA is unlabelled, θ_i is independent of the applied rule i and is equal to zero;

$$\theta_i = 0, \forall i : 0 \leq i < n.$$

From the above, we see that for every LMA an equivalent Turing Machine can be constructed, and similarly for every UMA an equivalent Turing Machine can be constructed.

On the other hand we can construct a Labelled or Unlabelled Markov Algorithm for every Turing Machine, as we have already shown.

Thus we can see that for every LMA we can construct an equivalent Turing Machine TM, and then we can construct an UMA equivalent to TM.

Since the construction of an equivalent LMA for every UMA is trivial (we modify every rule of LMA " $P \rightarrow Q;$ " to " $P \rightarrow Q, 0;$ ") we do not have to prove that result via a Turing Machine.

We thus have the following result:

THEOREM 4.1

For every LMA there exists an equivalent UMA and vice versa. That is, Labelled and Unlabelled Markov Algorithms are equivalent Grammatical Devices.

3. TURING MACHINES AND TYPE 0 GRAMMARS

Since Turing Machines and Markov Algorithms are equivalent Grammatical Devices (and as we have already mentioned, Grammars are restricted types of Markov Algorithms) we will try to construct a Type 0 Grammar (that is an unrestricted type of Grammar) equivalent to an arbitrary Turing Machine. The construction will be

the same as those of the corresponding constructions of the chapters I and III. If the Turing Machine is deterministic, respectively, nondeterministic the constructed equivalent Type 0 Grammar will be deterministic, respectively, nondeterministic.

This construction is possible, as we shall explain below, because although a Type 0 Grammar is a "slightly" restricted Markov Algorithm, there is nevertheless only one restriction on its productions. (And since this happens for deterministic and nondeterministic Type 0 Grammars, it is evident that even a deterministic Type 0 Grammar is a restricted deterministic Markov Algorithm). By definition if " $a \rightarrow b$ " is a production of a Type 0 Grammar, for a and b we have: $a \in A^+$ if the grammar is generative and $b \in A^+$ if the grammar is recognizing. In other words productions such as " $\Lambda \rightarrow b$ ", respectively, " $a \rightarrow \Lambda$ " cannot exist in a Generative, respectively, Recognizing Type 0 Grammar.

This however, does not pose any problem, because although a Turing Machine (Generative or Recognizing) can have sets of moves equivalent to the above productions (as can be seen in our construction of chapter I), it cannot have single moves equivalent to the same productions. On the other hand, in the construction of a Markov Algorithm equivalent to a given Turing Machine (chapter II), we included one MA rule for every TM move, and no rules of the type " $\Lambda \rightarrow b$ " or " $a \rightarrow \Lambda$ " were included in the set of rules of the constructed MA.

The same construction would be possible for a Markov Algorithm which is not allowed to have productions of the types " $\Lambda \rightarrow b$ " or " $a \rightarrow \Lambda$ ", as these productions would never be used for the simulation of TM under any conditions.

Thus we see that the Type 0 Grammars, although they are restricted

Markov Algorithms, are able to simulate exactly any Turing Machine. In other words we can say that "for every Turing Machine there is an equivalent Type 0 Grammar".

The converse, that is "for every Type 0 Grammar an equivalent Turing Machine can be constructed", is rather trivial, since the facts that a Type 0 Grammar is, as we have shown, a restricted Markov Algorithm and that we can construct an equivalent Turing Machine for every Markov Algorithm, cover this case.

Thus we have the following result:

THEOREM 4.2

Turing Machines and Type 0 Grammars are equivalent Grammatical Devices.

4. NONDETERMINISTIC AND DETERMINISTIC TYPE 0 GRAMMARS

As we have shown, for an arbitrary Nondeterministic Type 0 Grammar, NGO, an equivalent Nondeterministic Turing Machine, NTM, can be constructed. Since Deterministic and Nondeterministic Turing Machines are equivalent, we can construct a Deterministic Turing Machine, DTM, equivalent to NTM. Consequently a Deterministic Type 0 Grammar, DGO, can be constructed, equivalent to DTM and obviously to NGO.

On the other hand, it is trivial to construct a Nondeterministic Type 0 Grammar, equivalent to an arbitrary Deterministic one.

Thus we have proved the following result:

THEOREM 4.3

For every Nondeterministic Type 0 Grammar, an equivalent Deterministic Type 0 Grammar can be constructed and vice versa.

5. TYPE 0 GRAMMARS AND TYPE 0 PRODUCTIONS

The difference between a Type 0 and a Type 1 Grammar is that the former can include productions of the type: $a \rightarrow b$, where $a \in A^+$, $b \in A^*$ and $|a| > |b|$ if the Grammar is Generative, or $a \in A^*$, $b \in A^+$ and $|a| < |b|$ if the Grammar is Recognizing. We will call the above productions, which are length decreasing, for a Generative, and length increasing for a Recognizing Grammar, "Type 0 Productions".

Assuming now that we have an arbitrary Type 0 Recognizing Grammar RGO (deterministic or not) we can construct a Turing Machine, TM equivalent to RGO. In this construction for every Type 0 Production, there will be a set of TM moves, enabling TM to expand the string, on its tape, by printing X's on $b-a$ blank squares (at the right end of the string), and then by moving the printed X's to the left, so that the left most X will have to its left the square with the right most character of the string a . Then the string $\underbrace{ax \dots x}_{b-a \text{ X's}}$ will be substituted by b .

Constructing now another Grammar RGO', equivalent to TM, we can see that the only Type 0 productions will be all productions with the forms:

$$q_i B \rightarrow X q_j B$$

$$\xi q_i B \rightarrow q_j \xi X B;$$

where B (blank character) and X are special symbols in Γ and A , q_i and q_j are in \bar{K} and A , and ξ is in Γ and A .

Since the character B can only exist at the right end of the string, the above productions are applicable at this position only, and the Grammar RGO' including Type 0 productions is of Type 0.

RG0' is also equivalent to RG0.

Thus we have shown that for every Type 0 Grammar an equivalent Type 0 Grammar can be constructed with all its Type 0 Productions, applicable only at the right end of the string.

Although this proof is based on the constructions of the previous chapters, the result is not dependent on these particular constructions. Obviously, there may be other ways of constructing a Turing Machine equivalent to an arbitrary Markov Algorithm and vice versa. However in any of these different constructions, it will always be possible, when starting from a Markov Algorithm with Type 0 productions, to construct, via a Turing Machine, an equivalent Markov Algorithm such that, all its Type 0 Productions will be applicable only at the right end of the string. This can be explained as follows:

One of the reasons why a Turing Machine is an inflexible Device, (compared to a Markov Algorithm) is that it cannot modify (in this case increase) the length of the string on its tape, unless its head is scanning the first blank square of the tape. Thus a Markov Algorithm, constructed to simulate a Turing Machine, will behave, in this case, (as in any other case), as a Turing Machine. This means that the constructed Markov Algorithm cannot have rules of the forms:

$$q_i a \rightarrow b_2 b_1 q_j,$$

$$\xi q_j a \rightarrow q_j \xi b_1 b_2$$

where: $\xi, b_1 \in (\Gamma - \{B\})$, $b_2 \in (\Gamma - \{B\})^+$,

because this would imply that the Turing Machine should include corresponding (actually equivalent) single moves of the forms:

$$\delta(q_i, a) = (q_j, b_2 b_1, R) \quad \text{where} \quad b_1 \in (\Gamma - \{B\})$$

$$\delta(q_i, a) = (q_j, b_1 b_2, L) \quad b_2 \in (\Gamma - \{B\})^+,$$

which is a contradiction to a Turing Machine's definition.

On the other hand (after modifying every given string "w" to "q₀wB", before a Markov Algorithm starts the simulation of a Turing Machine) there must exist Markov Algorithm rules corresponding to the following Turing Machine's types of moves:

$$\begin{aligned} \delta(q_i, B) &= (q_i, b, R) \\ &\quad b \in \Gamma; q_i, q_j \in \bar{K}; \\ \delta(q_i, B) &= (q_j, b, L) \end{aligned}$$

The Markov Algorithm rules corresponding to the above moves are:

$$\begin{aligned} q_i B &\rightarrow X q_j B \\ \xi q_i B &\rightarrow q_j \xi X B, \quad X \in (\Gamma - \Sigma) \text{ being a special symbol.} \end{aligned}$$

As can be seen from the above rules we have

$$|LHS| < |RHS|,$$

which implies that they are Type 0 Productions.

This again means that if the constructed Markov Algorithm from a Turing Machine includes Type 0 Productions, they can be applicable only at the right end of the string.

Similarly it can be shown that for every Generating Grammar, an equivalent Generating Grammar can be constructed, such that all

its Type 0 Productions can be applicable only at the left end (that is at the beginning) of the string.¹

6. TURING MACHINES AND TYPE 3 GRAMMARS

Every type 3 Grammar, as indeed all types of Grammar, is also of type 0. Thus we can construct Turing Machines capable of accepting the Language generated by any type 3 Grammar. The fact that type 3 Grammars are restricted types of type 0 Grammars means that the corresponding Turing Machines will not have to use their entire power when they are to recognize strings generated by type 3 Grammars. In other words, if a Turing Machine, to be used exclusively as a recognizer for a type 3 Language, is to be constructed, it will be a restricted Turing Machine.

Obviously any Turing Machine of such a restricted type will not be able to recognize types of Languages, other than the ones generated by type 3 Grammars.

Now, every production of a type 3 Grammar must have the same

1 In the above section "left" and "right" are not absolute, in the sense that the word "left" can substitute the word "right" and vice versa.

What we have proved for the applicability of Type 0 Productions at the RIGHT respectively LEFT end of the string, can also be proved to be true, at the LEFT respectively RIGHT end of the string. This can be achieved if we use a Turing Machine with a two-way infinite tape.

Thus more generally we can say: For every Type 0 Grammar (Generative respectively Recognizing) an equivalent Type 0 Grammar (Generative respectively Recognizing) can be constructed such that all its Type 0 Productions can be applicable at either end of the string alone.

formation as one or other of the two following types of productions

$$A \rightarrow aB \quad (1)$$

$$C \rightarrow b \quad (2)$$

where $A, B, C \in V_N$ $a, b \in V_T$.

This means that at any time during the generation of a string, by a type 3 Grammar, there will be (as long as the generation has not been completed) only one variable in the string located always at the right end of the string. This variable may be substituted, each time, by the concatenation of a terminal and a variable which is achieved by the application of a type (1) production, or it may be substituted by a single terminal when the generation terminates; in this latter case a production of type (2) will be applied.

Thus the length of the string after each application of a type (1) production, is increased by one character, while it remains the same after the unique application of a type (2) production at the end of the string's generation.

It is also worthwhile to notice that in a generated string, every character present corresponds to the unique terminal found in the RHS of one or more productions.

6.1 Turing Machines recognizing Type 3 Languages.

Let us consider the arbitrary type 3 Grammar $G_3 = (V_N, V_T, P, S)$.

A Turing Machine TM_3 , needed especially as a recognizing device for the Language generated by G_3 , is defined as follows:

$TM_3 = (V_T, \bar{K}, \Gamma, \delta, q_S, F)$, where:

$\bar{K} \equiv (\{q_I / \forall I \in V_N\} \cup \{q_F\})$ is the set of states of TM_3

$\Gamma = (V_T \cup \{B\})$ is the set of all used characters

δ is a function from $K \times \Gamma$ into $K \times (\Gamma - \{B\}) \times \{R\}$, made up as follows:

For every G_3 production of the type $A \rightarrow aB(1)$ ¹, we include in δ the corresponding move:

$$\delta\{q_A, a\} = \{q_B, a, R\} ;$$

and for every G_3 production of the type $C \rightarrow b$ we include in δ the corresponding move:

$$\delta\{q_C, b\} = \{q_F, b, R\} .$$

No other moves are included in δ .

q_S is the initial state of TM_3

An automaton, like TM_3 , which reads characters in sequence (e.g. from a tape) cannot determine when a given string of characters has been entirely read. This happens because the automaton cannot detect whether or not the character just read is the last in the string, a thing which could be easily determined by a human being.

Actually a human being reading a text does not need to read a special symbol at the end of text as an indication that the text has been entirely read. The fact that there is nothing left to be read makes the human stop reading.

Thus for convenience only we can modify the function δ of TM_3 by including the additional move:

$$\delta\{q_F, B\} = \{q_F, J, R\} , \text{ (where } J \text{ can be any character in } V_T)$$

which is an artificial move. Having done that, we must also change the set of final states to $F = \{q_F\}$. In this case the set of states \bar{K} will be: $\bar{K}' = (\{q_I / \forall I \in V_N\} \cup \{q_F, q_F\})$.

F is the one element set of final states;

$F = \{q_F\}$ of TM_3 . (If the production $S \rightarrow \epsilon$ is included in G_3 then $F = \{q_F, q_S\}$).

6.2 A Turing Machine simulating a type 3 Recognizing Grammar

Just as in the case of all types of Grammars, for every type 3 Generating Grammar there exists its dual equivalent type 3 Recognizing Grammar. Thus the dual to the previously defined G_3 Generating Grammar is the Recognizing Grammar G_{3R} defined as follows:

$$G_{3R} = (V_N, V_T, P^{-1}, S)$$

where:

P^{-1} is the set of productions of G_{3R} such that:

- for every production with the form (1) in P , we include in P^{-1} the production $aB \rightarrow A(1)'$;
- for every production with the form (2) in P , we include in P^{-1} the production $b \rightarrow C(2)'$;
- no other productions are included in P^{-1} .

S is the ending symbol of G_{3R}

Since every production of a type 3 Recognizing Grammar is applicable only at the right end of the string, it is obvious that the recognition must start at this end and applications of the productions must take place from right to left.¹

1 "Right" and "left" in these cases are relative to the given definitions and if they are mutually transposed, nothing will have to be changed in the theory.

Thus a Turing Machine TM_{3R} constructed to simulate the Recognizing Grammar G_{3R} can be described as follows:

- a. The tape will be infinite to the left and the input string will be placed in such a way that its right most character will occupy the right most square of the tape.
- b. When the machine starts, its head will be scanning the right most square of the tape.

Formally TM_{3R} is defined as a system

$$TM_{3R} = (V_T, \bar{K}, \Gamma, \delta, q_0, F) \text{ where:}$$

V_T is the set of input symbols;

Γ is the set of all the allowed symbols,

$$\Gamma = (V_T \cup \{B\}) ;$$

$\bar{K} \equiv (\{q_I / \forall I \in V_N\} \cup \{q_0\})$ is the set of TM_{3R} 's states;

δ a function from $\bar{K} \times \Gamma$ into $\bar{K} \times (\Gamma - \{B\}) \times \{L\}$.

The moves included in δ are:

$\delta\{q_B, a\} = \{q_A, a, L\}$ for every production of the type $aB \rightarrow A$ in P^{-1} and

$\delta\{q_0, b\} = \{q_c, b, L\}$ for every production of the type $b \rightarrow C$ in P^{-1} ;

q_0 is the initial state of TM_3 ;

$F = \{q_S\}$ the one element set of final states.

6.3 Conclusion

From the above we can finally notice that the constructed TM_3 and TM_{3R} Turing Machines, both designed exclusively for the re-

cognition of the arbitrary type 3 Language L_3 , are restricted types of Turing Machines, because:

- a. their heads are able to move in one direction only (i.e. left or right)
- b. their tapes are bounded. That is their heads cannot surpass the first blank square, next to the one with the last (right most for TM_3 and left most for TM_{3R}) character of the string.

7. TURING MACHINES AND TYPE 1 GRAMMARS

The type 3 Grammars, just as the type 1 ones, are restricted types of type 0 Grammars. In fact every type 1 Grammar is a type 0 Grammar without "Type 0" productions.

Thus a Turing Machine used exclusively as a recognizer of a type 1 Language, will be of restricted type. It is obvious that any Turing Machine of this type will be able to recognize any Languages generated by type 2 and type 3 Grammars, which of course are type 1 Grammars as well.

Every type 1 Generative Grammar is defined so that for every of its productions " $a \rightarrow b$ " we must have: $|a| \leq |b|$. This means that after every application of any one of the productions of a type 1 Generative Grammar, the length of the string under generation may either remain the same or be increased. Thus if S is the start symbol of a Generative type 1 Grammar and Ψ the generated string, we will have:

$$|S| \leq |\Psi| \quad ,$$

and since $|S| = 1 \leadsto |\Psi| \geq 1$.

7.1 Turing Machines Recognizing Type 1 Languages

For every type 1 Generative Grammar, as well as for all other types of Generative Grammars, there exists a dual equivalent, type 1 Recognizing Grammar, which accepts the language generated by the Generative one. If now by $G_{1G} = (V_N, V_T, P, S)$ we define a Generative type 1 Grammar, its dual recognizing G_{1R} , will be defined by $G_{1R} = (V_N, V_T, P^{-1}, S)$.

Since the Grammar G_{1R} is a restricted type of Markov Algorithm (as is any Grammar) we can construct a Turing Machine TM_1 , to simulate the Grammar G_{1R} , in such a way that TM_1 will be able to do exactly what G_{1R} can do and nothing more. Then TM_1 will, obviously, be a recognizing device for the Language generated by the Grammar G_{1G} .

The construction of TM_1 will be similar to the construction in section 3 of the present chapter, where we examined the relations between Turing Machines and type 0 Grammars.

Since now the difference between type 0 and type 1 Grammars is that the latter does not have "Type 0" productions, the Turing Machine TM_1 has to be constructed in such a way that it will not be able to accept this kind of productions.

For every production of the Grammar G_{1R} we must have

$$|lhs| > |rhs| .$$

Thus it is obvious that the tape of TM_1 will never have to be expanded, during the examination of any input string, generated by the Grammar G_{1G} . This means that the head of TM_1 will never

scan the second blank square of the tape¹ for any string of this type. From this we understand that the tape of TM_1 does not have to be infinite in either direction (that is left or right), but it will have a specific length as we see below.

The fact that $|S| \leq |\Psi|$, where S is the start symbol for G_{1G} and the ending symbol for G_{1R} and Ψ is a generated string by G_{1G} , (the one which will be recognized by TM_1), means that the length of the tape will be a function of $|\Psi|$, dependent on the particular construction of TM_1 .² For example, in a construction similar to that of chapter I we will have:

$$\text{length of the tape} = |\Psi| + 2;$$

in this case the tape of TM_1 , after the string Ψ has been placed in it, will have the form:

$$S \ \Psi \ B \ .$$

Thus we see that the tape of a Turing Machine of this type, will be of length $|w| + 2$, for any string w which is to be recognized.

- 1 We say the second and not the first blank square of the tape, because the first one may be scanned by the head in some cases, exclusively with the view of getting the information "where the string finishes", as this cannot be detected alternatively. However, whenever this happens, the square in question (which will be always the same), will remain blank, as we have already explained in the construction of chapter I, (section 4.1).
- 2 As it is obvious, in no case the length of the tape can be less than $|\Psi|$ and there is no need to be greater than $|\Psi| + 2$.

CHAPTER V

COMPUTER PROGRAMS FOR DEMONSTRATING THE EQUIVALENCE OF TURING MACHINES AND MARKOV ALGORITHMS

1. INTRODUCTION

In this chapter we shall concern ourselves with detailing the application of the constructions of chapters I and II by suitable computer programs.

Thus we have written a Turing Machine and a Markov Algorithm simulators and two other programs, the one producing a Markov Algorithm simulator equivalent to a given Turing Machine and the other producing a Turing Machine simulator equivalent to a given Markov Algorithm.

The programs are written in C language and executed in a PDP 11 computer under the UNIX operating system.

2. A TURING MACHINE SIMULATOR

This simulator is theoretically able to do exactly what a Basic Turing Machine Model can do. We use the word "theoretically" only in view of the space and time limitation found in any contemporary computer.

A Turing Machine, as an abstract machine is independent of restrictions related to the space or time needed for the execution of a program. This means that not every Turing Machine can be simulated by a computer program, but for all cases where this is possible, the Turing Machine Simulator Program will behave exactly as its abstract prototype.

2.1 Representation of a Turing Machine in a Computer

The set of all characters used by the machine is stored in a character array called Alphab. Then each character is represented by the number of the cell it occupies; that is by an integer.

Another array of integer type this time, called tape simulates

the tape of the machine. Two, two-dimensional integer arrays, called print and state, represent the function δ . The former determines the character "to be typed" and the second the "next state" for every state-character pair of a defined move. From the array state it can be also determined if the next move of the head will be to the left or to the right. This is achieved by storing a negative integer for the "left" moves, for any state-character pair. Finally a pointer pointing to the square "presently scanned" by the head, is called square and it symbolizes the head of the machine.

2.2 Input - output

An input Turing Machine program, given to the simulator, will be of the following form.

The first line¹ of input, will be a string of characters, namely the characters in Γ , excepting the character B. It has been arranged that B always occupies the first cell of the array Alphab. The second line of input is the input string to the Turing Machine. The moves of the Turing Machine occupy the next lines of input, in such a way that every move is written as an input line. Every move is given as a quintuple of the form

$$q_i \quad a \quad b \quad q_j \quad R$$

which is the representation corresponding to

$$\delta(q_i, a) = (q_j, b, R) .$$

1 "Line" here has the meaning of an uninterrupted set of input which is independent of the "length" of characters in a line of any computer terminal (or card).

Finally in the last lines of input, the word "OPTIONS" is followed by a line of options for the desired type of output. If however options are not specified, the default options are set automatically. There are eight different options for output. They are useful when a Turing Machine program is very long and we wish to check the tape in a number of respects like checking the condition of the tape at certain intervals comprising a specified number of moves, or at certain otherwise specified "special" moves, or at each one move belonging to a number of successively executed moves, or finally checking how the tape is just before halting.

In every line of output there is firstly the move under execution and then a representation of the tape as it is just before this execution.

2.3 Subroutines Used

Two small procedures are used for setting up the data. These are the procedure called "initial" setting up the arrays Alphab and tape and the procedure called delta setting up the arrays print and state, that is the function δ .

A routine called figure prints a line of output whenever is called. Finally the procedure called showtape determines, according to the options given, when figure will be called.

2.4 A Version with double-subscript states

By some modifications in the program, it was arranged that every state previously represented by q_K in the input, may be represented in this version by $Q(i,j)$. This will enable the Turing Machine

Program to simulate a Markov Algorithm, in a way similar to the one of Chapter I.

3. A MARKOV ALGORITHM SIMULATOR

As is known Markov Algorithms, although equivalent to Turing Machines are much faster and sophisticated devices. This has as a result that the internal function of a Turing Machine is much simpler than that of a Markov Algorithm. Thus a computer program simulating a Markov Algorithm has to be more complicated than a program which simulates a Turing Machine.

As in the case of the Turing Machine simulator, a Markov Algorithm one will be able to behave as its abstract Markov Algorithm counterpart whenever space and time problems do not present an obstacle to the computer.

The Markov Algorithm simulator, described in the following pages will later be used to simulate a Turing Machine, by a suitable transformation of a Turing Machine program to an equivalent Markov Algorithm one.

The program is designed so that any (theoretically) number of alphabets may be used and any number of generic variables may correspond to each alphabet. It will also be capable of accepting productions with LHS's and for RHS's containing symbols belonging to input alphabets (even if generics are used) including cases where the priority of substitution is important.

3.1 Input to the Program

Every input to the Markov Algorithm simulator will be of the following form.

First there will be a number of input lines equal to the number of alphabets used by the algorithm. Every one of these lines will consist of the characters of one of the alphabets followed by the generics corresponding to this alphabet. Then the input lines with the rules of the algorithm follow, and finally the input string followed by one line of options.

Examples of input

$$a_{11}a_{12}\dots a_{1K_1}/\xi_{11}\xi_{12}\dots\xi_{1\lambda};$$

$$a_{21}a_{22}\dots a_{2K_2}/\xi_{21}\xi_{22}\dots\xi_{2\lambda_2}$$

.

.

.

$$a_{m1}a_{m2}\dots a_{mK_m}/\xi_{m1}\xi_{m2}\dots\xi_{m\lambda_m};$$

n

$lhs_1 \rightarrow rhs_1, i_1;$ or $lhs_1 \rightarrow rhs_i,$ if the algorithm is
unlabelled

$lhs_2 \rightarrow rhs_2, i_2;$

.

.

$lhs_n \rightarrow rhs_n, i_n;$

"input string"

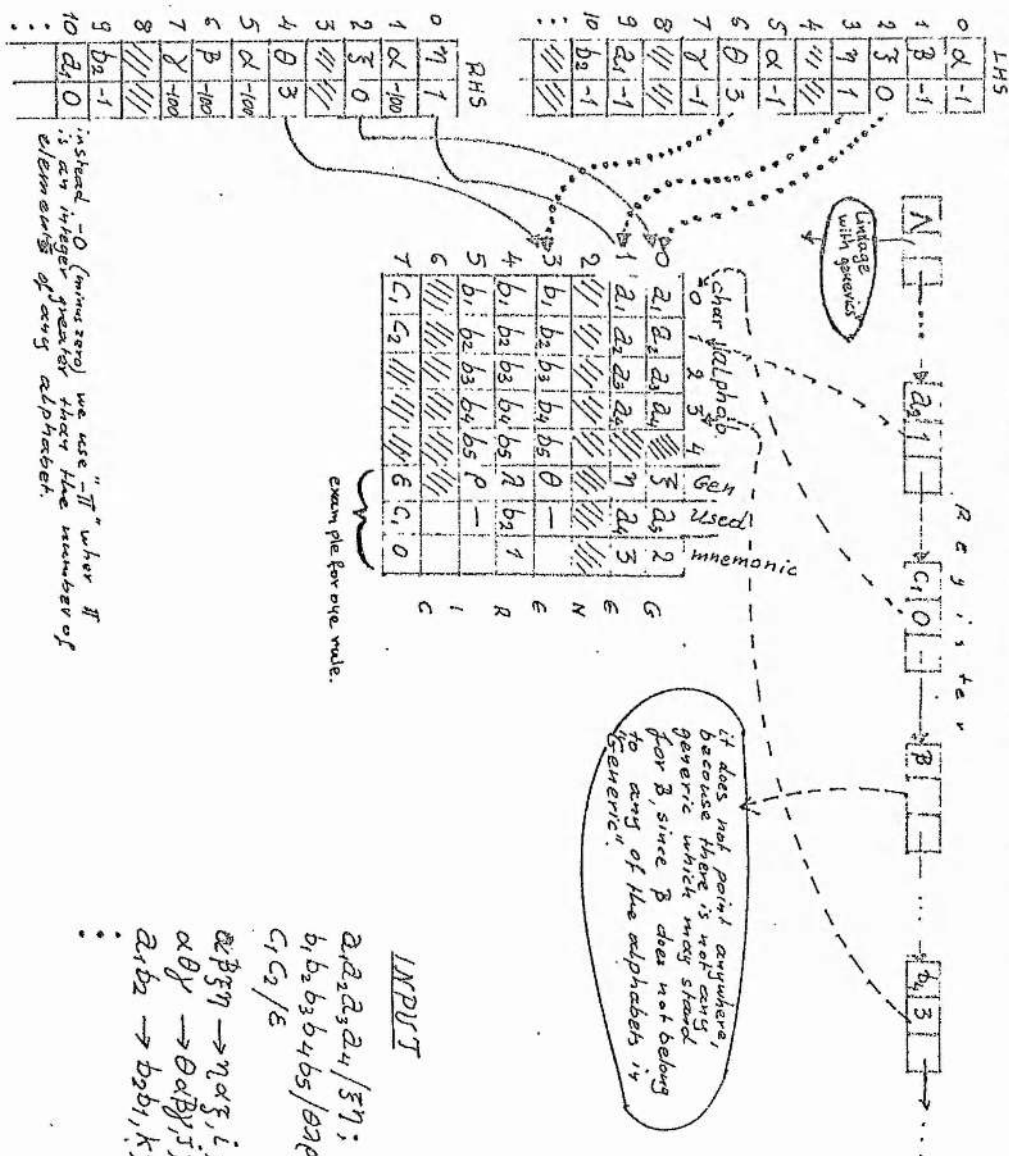
options.

3.2 Data Structure

A very useful property of the C language is the use of structures which help greatly to the program's readability. Thus we use the following array structures.

Rules	0	1	2	3	...	n-1
l _r point	0	5	9			
l _p point	4	3	2			
diff	4	1	7			
r _p point	0	4	9			
label	L	J	K			

It determines the cell of LHS_i where is the first char. of the rhs of rule i.
 The length of LHS_i:
 $diff = |RHS_i| - |LHS_i|$
 It points to the cell of RHS_i where the first character of rhs_i is.
 It determines the next rule to be tested.



- a. LHS with two fields; the first, called Lsym, is of character type and contains the lhs's of all rules sequentially, by separating every last character of a lhs of a rule from the first character of the lhs of the next rule by a special symbol. In the second field called Lgen, and being of integer type, there is an integer describing the character in the corresponding field of the same "line" of the structure; that is it determines if it is an input symbol or a generic or a marker.
- b. RHS is a structure similar to LHS.
- c. Another structure called Rules uses five fields of integer type. Thus Lpoint points to the cell of LHS containing the first symbol of the lhs of the corresponding rule. In the field called Lplatos there is an integer (for every rule) equal to the length of the lhs of the corresponding rule. The difference in length between lhs and rhs of a rule is stored in a field called diff. Finally an integer determining the next rule to be tested is stored in the field called Label.
- d. The array structure Generic is more complicated since it has every line consisting of
 - (i) an array field, of character type called alphab, storing one of the input alphabets;
 - (ii) a single character field where the generics corresponding to alphab are stored;

- (iii) a character field called used indicating each time the character into which the generic was translated;
 - (iv) the field mnenonic, which will be explained later.
- e. The structure Register is a three-field structure, where, in every one of its first character-type field a character of the string is stored, in every second field an integer determining the column of the character-array field alphab of the structure Generic where this character can be found. The third field, of integer type, is a pointer to the next element of the structure. When the program starts, a number of elements of the structure equal to the length of the input string is used. All the unused elements form a stack; a pointer called AVAIL points to the first (top of the stack) unused element. Thus whenever one or more new elements are needed from the stack, the one pointed to by AVAIL is always taken as first, and the second becomes that pointed to by AVAIL and so on. On the other hand every element used and not needed any more is placed on the top of the stack and is pointed to by AVAIL. The first of the used elements of Register is always occupied by the character "A" (that is the empty string).

A data structure diagram occupies the following page.

3.3 Procedures used

The procedures Initgen, Initrule, Introduce and Debbuging set

up the data, reading from input.

The procedure search is designed to perform the applicability search for every rule being tested and is called only when the lhs of the rule is not empty. Thus "search" compares, character by character, the string found in the array Register and the lhs of the rule being tested, looking for a first appearance of the lhs in the string. By calling a small procedure called anymore, search detects if the length of the part of the string not yet examined is at least equal to the length of the lhs of the tested rule, whenever the first character of lhs is compared with a character of the string. It is also examined by the help of another small procedure, called carryon, if there is a next character in the string. If there is not another character in the string anymore is not used.

When a character of the string is compared to a symbol in LHS, the following are possible:

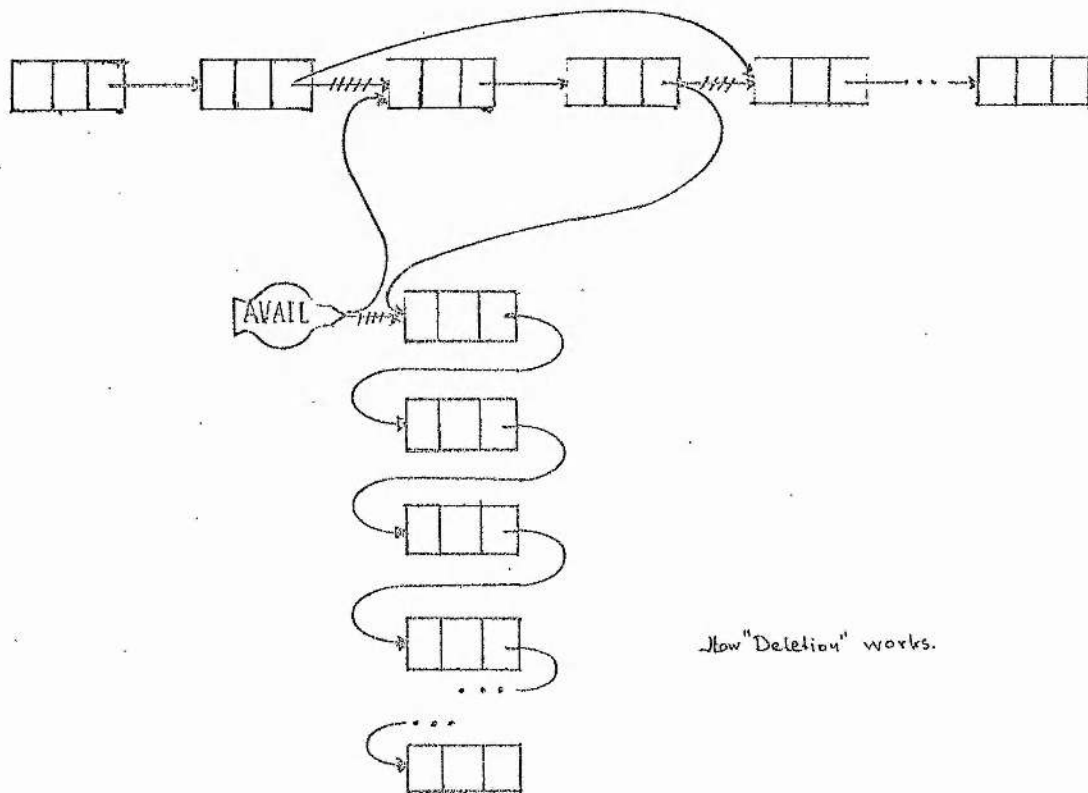
1. The character in the string is the same as the one in LHS (which means that the symbol in LHS is not a generic).
- or 2. The symbol in LHS (field Lsymb) is a generic and the character in the string (Register) belongs to one of the alphabets (that is, it is not a marker), which is the range of the generic variable. In this case there are also two possibilities examined; that is whether the generic has been used before in this rule or it has not. This is specified by the fields of generic mnemonic.
3. If none of cases 1 or 2 happen the rule is inapplicable.

If a rule is applicable "search" returns "1", and "0" otherwise.

3.3.1 Procedure Apply

This procedure is called whenever $|lhs| = 0$ or when search returns the value 1. Two small procedures Insertion or Deletion are used by Apply when the lengths of lhs and rhs are unequal.

Thus when $lhs > rhs$ the procedure deletion is called which removes a number of unwanted elements of the used part of Register (string) by joining the last one (right most) to that pointed at by AVAIL in the free list (stack) and by changing the value of AVAIL so that the first (left most) element is pointed at by AVAIL. It also changes the pointer value of the element to the left of the left most unwanted one in the used part of Register (string) so that it now points to the element pointed at previously by the right most unwanted element.



How "Deletion" works.

The procedure insertion is used when $lhs < rhs$ and is the "inverse" of deletion.

If $lhs = rhs$ or after deletion or insertion has been called for a rule by "apply" the substitution of LHS by RHS takes place.

3.4 Subscript Version

As we have mentioned, the program, by some small modifications and additions (e.g. a routine called modify), is able to accept rules with markers using subscripts of the form $Q(i)$ and thus it can simulate a Turing Machine with (theoretically) any number of states $Q(i)$, after a suitable transformation of the Turing Machine Program.

4. A PROGRAM PRODUCING A TURING MACHINE EQUIVALENT TO A GIVEN MARKOV ALGORITHM

This program which is based on the construction of chapter I, reads from its input the description of a Markov Algorithm and produces a Turing Machine which can simulate exactly the Markov Algorithm. The input to this program is identical to the input to the Markov Algorithm Simulator.

Any type of Markov Algorithms can be accepted as input; thus a Markov Algorithm with generic variables and with one or more Alphabets and with any (practically possible) number of generic variables for every Alphabet.

4.1 The Problem of Generics

In this section we will refer to cases of Markov Algorithms which use more than one alphabet and a number of Generic variables for every alphabet.

We suppose then that a Markov Algorithm M_a has n alphabets,

A_1, A_2, \dots, A_n which are disjoint and their union is a subset of the alphabet A , which is the alphabet of all used symbols.

We also suppose that for a number m of alphabets, where $m < n$, there are m sets of generic variables, that is one set of generics for every one of the m Alphabets.

We also assume that every alphabet A_j has K_j elements $V_j : 1 \leq j \leq m$. If now to an alphabet A_p , with K_p elements, there correspond r_p generic variables, we will represent the latter by $g_{p1}, g_{p2}, \dots, g_{pr}$

Thus we have the following table:

Alphabet	number of elements	Generics	number of generics
A_1	K_1	$g_{1_1}, g_{1_2}, \dots, g_{1_{r_1}}$	r_1
A_2	K_2	$g_{2_1}, g_{2_2}, \dots, g_{2_{r_2}}$	r_2
\vdots			
A_m	K_m	$g_{m_1}, g_{m_2}, \dots, g_{m_{r_m}}$	r_m

Finally with $g(A_j, q)$ where $1 \leq j \leq m$, $1 \leq q \leq r_j$ we symbolize the q th generic of the alphabet A_j .

Now we suppose that the Turing Machine T_m which simulates the Markov Algorithm M_a , is working in the i th M_a rule, trying to find a first occurrence of the lhs_i on its tape. We assume that

$$\text{lhs}_i = "a_1 a_2 \dots a_r"$$

where $a_i \in (A \cup \{\text{generic variables}\})$, $\forall i : 1 \leq i \leq r$. We also assume that all a_i , $i = 1, 2, \dots, s-1$, where $s < r$, have been found in

successive squares of the tape and that T_m being in state $Q(i,t)$ is looking for an a_s in the next square to the right. Then we have two cases to examine.

a. $a_s \in A$

In this case if the character a_s is found in the next square to the right, in the tape, the following move has to be executed,

$$\delta\{Q(i,t), a_s\} = \{Q(i,t+1), a_s, R\}$$

But if the character in the next square is a character $\xi_s \in (A \cup \{B\} - \{a_s\})$, then the move

$$\delta\{Q(i,t), \xi_s\} = \{Q(i, u_f), \xi_s, L\}$$

has to be executed, where $Q(i, u_f)$ are states used, whenever during the applicability search for the rule being tested, a symbol read from the tape does not match with the presently tested symbol in the lhs.¹ Thus we have a situation indicated in the diagram 1.

1 We use the subscript f in state $Q(i, u_f)$ because the second subscript u_f will be different for different cases. E.g. a "B" is scanned before the whole lhs has been found on the tape, in which case the rule is inapplicable.

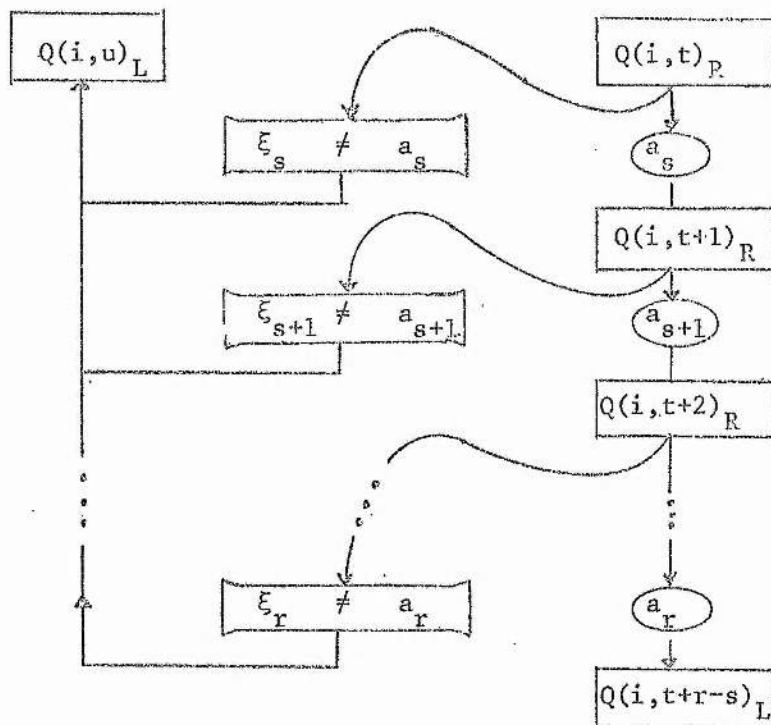


Diagram 1

b. a_s is the generic variable $g(A_j, q)$;

that is $\text{lhs} = "a_1 a_2 \dots a_{s-1} g(A_j, q) a_{s+1} \dots a_r"$.

If the alphabet A_j has k_j elements, this means that k_j next moves and k_j next states are needed. Obviously depending on the character which will be read from the tape, only one of these moves will be executed and only one of these states will be used by T_m , at this instance.

Thus if $A_j = \{a_{j_1}, a_{j_2}, \dots, a_{j_{k_j}}\}$, it is possible that one of the characters in A_j may occupy the next square to the right of the tape. Since then any of these characters matches with the generic $g(A_j, q)$ in the lhs of the examined rule, there must be a move and a next T_m 's state for every character in A_j . If T_m is

in the state $Q(i,t)$ before it has scanned the next square to the right, its possible next states will be:

$$\{Q(i,t+1), Q(i,t+2), \dots, Q(i,t+K_j)\}$$

The moves leading to the above states are:

$$\delta\{Q(i,t), a_{j_1}\} = \{Q(i,t+1), a_{j_1}, R\}$$

$$\delta\{Q(i,t), a_{j_2}\} = \{Q(i,t+2), a_{j_2}, R\}$$

.

.

.

$$\delta\{Q(i,t), a_{j_{K_j}}\} = \{Q(i,t+K_j), a_{j_{K_j}}, R\}$$

In case the scanned character is $\xi_j \notin A_j$ we also need the move:

$$\delta\{Q(i,t), \xi_j\} = \{Q(i, u_f), \xi_j, L\}$$

This situation can be seen in the diagram 2.

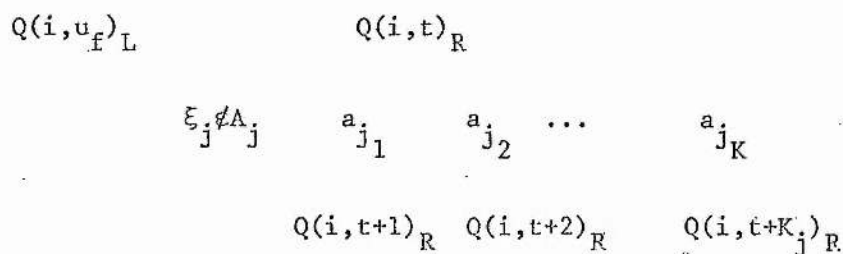


Diagram 2

If now the next symbol in lhs is a generic variable, $g(A_j^!, q')$, for each one of the above moves and states, a number of $K_j^!$ next moves and $K_j^!$ next states are needed, where $K_j^!$ is the number

of elements in alphabet A_j' . For each of the above moves a move leading to the state $Q(i, u_f)$ is needed.

Thus we see that we need $K_j \times K_j'$ direct successor states for the state $Q(i, t)$.

In the case, however, where the next symbol in lhs belongs to alphabet A (i.e. it is not a generic), only one next move and one next state is needed. The same happens if the next symbol in lhs is a generic which has already appeared in lhs before; in this case this generic will be treated as if it was a simple character.

We suppose now that the lhs of the rule i is constituted of r symbols which may be simple characters or generics. We may assume that the generics in lhs do not have repetitions; because if there are repetitions for some of them, each of these repetitions will be treated as a simple symbol (the same symbol into which the generic was "translated" in its first appearance). Thus if we start with a Tm's state, e.g. $Q(i, t)$, we can design a tree with the state $Q(i, t)$ in its root. (Diagram 3). Every node of the tree may represent a state or a simple character or a generic; the nodes at the same level are of the same kind. In the following example we symbolize by "sym" every node representing a simple character and by "gen, j " every node which represents a generic variable with range an alphabet with j elements. With "other" we symbolize the nodes representing all the "other" characters, that is the ones which are different to "sym" or do not belong to the alphabet A_j which is the range of

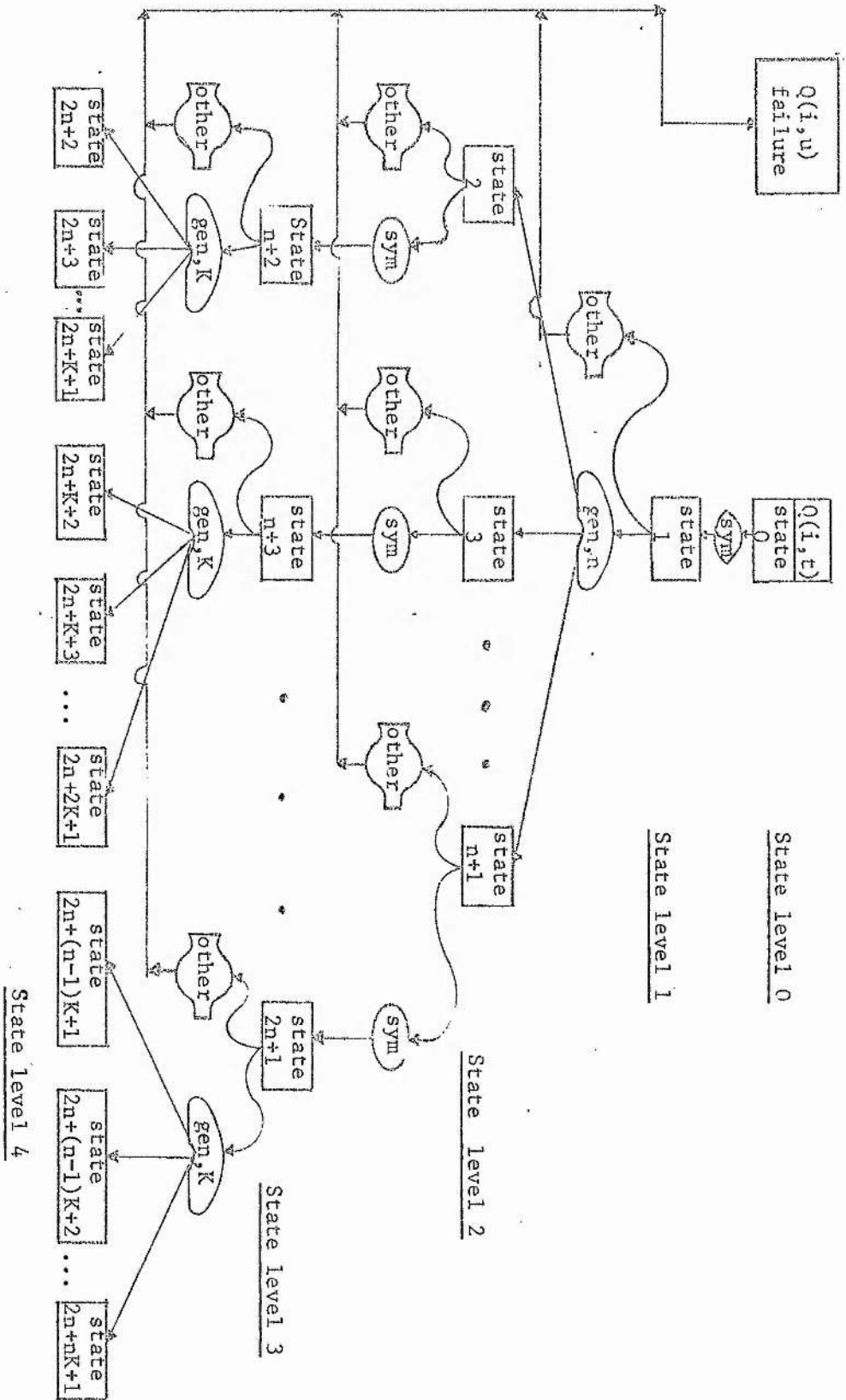


Diagram 3

the generic " $g(A_j, q)$ " (represented by the node " gen, j ").¹

4.2 Data Structure

The internal representation of characters is achieved by using an integer array Alphabet where all the simple characters (not generics) are stored. Thus any character in Alphabet may be represented by the number of the cell it occupies.

An example of arrays Alphabet and Bound

		Bound			
		Lower		Upper	
0	B markers used by T_m				
1	X	0	→	→	
2	Y	1	6	8	ξ
3	\neq	2	6	8	η
4	S	3	6	8	ρ
5	Λ	4	9	11	θ
6	a alphabet with generic variables ξ, n, p	5	9	11	π
7	b	6	12	13	μ
8	c	7	12	13	ν
9	K alphabet with generic variables θ, π				
10	l				
11	m				
12	t alphabet with generic variables μ, ν				
13	p				
14	a_1 markers appearing in the rules of M_a				
15	a_2				
16	a_3				
17	a_4				
18					

numsym = 18

1 Diagram 3, despite its appearance to the contrary, is really a tree, since $Q(i, u)$ is the node following upon every single other node and thus should have been signaled after each one of them; its single occurrence in the above diagram is only a matter of convenience.

Thus, as we can see in our example, we can divide Alphabet into smaller parts, so that while the first contains the markers used by Tm only, the next parts are occupied by the characters of the several alphabets (if the alphabets are more than one), and the markers appearing in the rules of Ma occupy the last part. No generics are stored in this array. By using however a structure called bound with two integer fields lower and upper we can determine, in Alphabet, the range of every generic. This is achieved by storing in the fields lower and upper, corresponding to a generic, the integers representing the first and the last, respectively, characters (as they are stored in array Alphabet) of the alphabet to which the generic corresponds.

The representation of the generics by integers is achieved by storing them in the field var of a structure called Generic which has another two integer fields called pos (position) and used. The usage of these two fields is to be explained later.

For the Turing Machine construction we need other markers in addition to the ones occupying the first six cells of the array Alphabet, in our example. These will be of four different types and will be all symbolized by T(i). An integer is enough for their internal representation, as we will show. Thus we use a variable numsym which determines the number of characters in array Alphabet. Maxdif is also an integer equal to the maximum of $-(|LHS_i| - |RHS_i|)$, for all rules. We also use the integers Remi, Jmemo such that:

$$Remi = maxdif + numsym;$$

$$Jmemo = Remi + numrule + 2;$$

where numrule is the number of rules of M_a .

Now we are ready to explain the four types of the $T_{(i)}$ markers.

1. $T_{(i)}$ is a type of marker used as a special symbol on the square of the tape, which is to the left of the one with the first character of the string, during the applicability search and application of a rule i . It has thus exactly the same meaning as the corresponding T_i of the construction of chapter I. The internal representation of $T_{(i)}$ will be an integer i' such that:

$$i' = \text{Rem}i + i.$$

For i' it will also be: $\text{Rem}i \leq i' < j\text{memo}$

2. $T_{(d)}$ will be a type of marker with the same meaning as the markers D_p of section 4.8.2.1 of chapter I; that is they determine the number of squares with X's which will have to be added to the right of every D_p , during the stage of the tape's expansion. The internal representation of a $T_{(d)}$ will be d' , where:

$$d' = d + \text{numsym}.$$

For d' it will also be:

$$\text{numsym} \leq d' < \text{Rem}i$$

3. $T_{(j)}$ is another type of marker used whenever the tape has to be expanded, in order to remind T_m of the second subscript j of its last state $Q(i,j)$, in which T_m was exactly before it started the expansion of the tape. This happens because as in the corresponding case of chapter I, there are special states (reserved) used during

the executions of the routines "expansion" and cut off".¹

For the integer j' representing the marker $T_{(j)}$ we have:

$$j_{\text{memo}} \leq j'.$$

4. As in chapter I a marker $T_{(s)}$ is used when during the applicability search for a rule i the first symbol in lhs_i matches a symbol read from the currently scanned square of the tape. Then the marker $T_{(s)}$ is printed in order to remind T_m of this square. Thus if the character found in this square is represented by s' , we represent $T_{(s)}$ with the integer $s + \text{con}$; we chose con big enough so that $\forall j$ representing the marker $T_{(j)}$ we have: $j' < \text{con}$. For the previous marker $T_{(j)}$ we have:

$$j_{\text{memo}} \leq j' < \text{con}.$$

The rules of M_a are stored sequentially in two one-dimensional integer arrays LHS and RHS. In fact the integers which represent the characters in LHS and RHS of each rule are stored in these arrays. Every generic is represented in LHS and RHS by a negative integer. For this reason the first cells (that is the zero cells) of the array structure Generic and array bound are not used.

An array structure called Rules with length equal to the number of rules of M_a has five integer fields; thus the fields L point and R point contain for every rule of M_a pointers to the first elements of the left and right hand sides, in arrays LHS and RHS. The fields L platos and R platos contain the lengths

1 Although in this program the existence of these two T_m sub-routines cannot be seen, their moves similar to the corresponding ones of chapter I are generated by the procedures Expand and Contract.

of the left and right hand side respectively. In the field dif is stored the difference $|lhs_i| - |rhs_i|$ for the corresponding rule i , and in field lab the label of the next rule to be tested.

We also use some other arrays which will be described later.

4.3 The Program

Although the program is complicated, we tried to maintain readability. Thus the program has been divided into small tasks performed by several procedures, and so it is very clear what each procedure does. The main part of the program after calling the three initializing procedures classify, initrule and initrest generates the very first move, by calling the procedure makemove, which has seven parameters; these are the two subscripts of the "present" (for the move) state, two integers representing the characters read and typed, respectively, the two subscripts of the next state and a character 'R' or 'L' with obvious meaning.

Then in a loop, every rule of M_a is examined and the corresponding T_m moves are generated, by calling the suitable procedures.

Thus for the rule i under examination it is first tested if $|lhs_i| = 0$, and if it is, the procedure Leftempty is called. If however $|lhs_i| > 0$, then after the procedures prepare and filltable, the procedure firstmoves which generates the first moves and the moves used by all rules is called. Then, if there is more than one character in lhs the procedure nextmove is called. The procedures caseone and casetwo are used for the generation of the moves necessary for the application of a rule found

applicable. The former generates the moves for the rules with equal lhs and rhs and for rules with $|\text{lhs}| > |\text{rhs}|$ and $|\text{lhs}| > 1$. The latter generates the moves for all rules with $|\text{lhs}| < |\text{rhs}|$ where $|\text{lhs}| > 1$.

Finally the procedures expand and contract which generate the moves for the expansion and "cut-off" of the tape, respectively, are called, if necessary.

In the following sections we will describe and analyse the most important procedures used in the program and refer to some techniques used in solving the several problems of the program.

4.4 Generics; next states and next moves

For the generation of next moves and the creation of next states of T_m corresponding to a rule of M_a including generics in its lhs or in both sides, we follow the technique described below.

As soon as the initial moves, corresponding to a rule i , have been generated, we use the variables jstart and jlast to represent the left most and right most, respectively, states (already created) of the same level, in the tree described in the section about generic variables. If we use as an example diagram 3 of the same section, and supposing that the last states generated are the ones of level 4, we will have

$$\begin{aligned} \text{jstart} &= \text{state } 2n+2 \\ \text{jlast} &= \text{state } 2n+nK+1. \end{aligned}$$

If now the next symbol to be examined in the lhs of the currently examined rule is a simple character, we must have $\text{LHS}[\text{L pointer}] > 0$; but if it is a generic we will have $\text{LHS}[\text{L pointer}] < 0$

and $gen = -LHS[L\ pointer]$; thus in the following parts of the program we generate the necessary next moves and create the next states:

1. Supposing that $readsym = LHS[L\ pointer] > 0$.

```

s = 6;
while (s < readsym)
{
    for (jvar = jstart; jvar < jlast; jvar ++)
        makemove (i, jvar, s, s, i, uf, L);      A
    s++;
}
s++;
while (s < numsym)
{
    repeat A
    s++
}
jincr = 1
for (jvar = jstart; jvar < jlast; jvar ++)
{
    makemove (i, jvar, readsym, readsym, i, jlast+jincr, R)
    jincr++;
}

```

2. If $LHS[L\ pointer] < 0$.

In this case we examine if this generic has been already used, (that is if it also appears (in lhs) to the left of the position in which it is currently examined). This can be easily detected if

there is "1" in the generic field "used" of the array structure Generic. (There will be a zero otherwise). If the generic has been used before, it will be treated in this case as a simple symbol. Thus we suppose that this is the first appearance of this generic in the lhs of the rule under examination. As we have said this generic may represent a part of the array alphabet (we mean here some characters occupying successive cells in this array).

Thus we have: s = gen.

```

s = 6;
while (s < bound [gen] .lower)
{
  for (jvar = jstart; jvar < jlast; jvar ++)
    makemove (i,jvar,s,s,i,uf,L);
  s++;
}
s = bound [ gen ] .upper+1;
while (s ≤ numsym)
{
  repeat (A)
  s++;
}

```

(A)

```

s = bound [ gen] .lower;
jincr = 1;
while (s < bound [ gen] .upper)
{
  for (jvar = jstart; jvar++; jvar < jlast)
  {
    makemove (i,jvar,s,s,i,jlast + jincr,R);
    jincr ++;
  }
  s++;
}

```

4.5 Procedure Leftempty

This procedure is called whenever the lhs of the rule under examination is empty. Obviously no generics exist in this case. If it also happens that $|rhs| = 0$, the suitable moves are generated for the preparation of T_m to start the applicability search for the next rule to be tested.

If however $|rhs| > 0$, Leftempty arranges for the generation of the moves needed for the expansion of the tape. The procedure expand, that is the one which generates the necessary moves for the expansion of the tape in all the other cases, is not used in the present case. This happens because the absence of generics in this particular case ($|lhs| = 0$), as indeed the absence of any character in lhs, makes the generation of the necessary moves easier. Thus Leftempty generates the moves which add a number $r = |rhs|$ of squares with Xs to the right of

the square with the characteristic symbol T_i of the rule under examination, and finally substitutes the X's by the rhs of the rule.

4.6 Procedure Prepare

This procedure uses an integer array called place where it stores sequentially every generic appearing in the lhs of the currently examined MA rule. It is finally arranged that only the left most appearance of each generic in lhs is kept in place, and every repetition is deleted (replaced by zero).

Then the generics left in place are stored in the corresponding field gener of the array structure Position. This is a structure with three integer fields gener, cardinal, step, having the following meaning. First the number of lines (elements) of this structure used for any rule under examination is equal to the number of distinct generics found in the lhs of the rule in question. Thus the first generic in lhs occupies the field gener of line 1 of the structure (element, or line, zero is not used), the second generic occupies the field gener of the line 2 and so on. The field cardinal determines the number of elements of the alphabet being the range of the corresponding generic.

Also the number of each cell of place where a generic is stored, (the distinct one in lhs only) is stored in the field pos of the corresponding line of structure Generic. Thus for every generic appearing in the lhs of the rule under examination there is an integer, greater than 0 stored in its corresponding field pos in structure Generic, which determines the position

of the generic in the lhs of the rule (relative to the other generics, if any, and not to simple symbols).

"Prepare", produces also for every rule a useful integer, called gencomb. This number is equal to the product of the cardinal numbers of the alphabets which are the ranges of the generics appearing in the lhs of the rule.

Finally "prepare" produces integers stored in the fields step of the array structure position as follows,
 $\text{position}[i].\text{step} = \text{position}[i-1].\text{cardinal} * \text{position}[i-1].\text{step};$
 the meaning of step will be explained in the next section.

4.7 Procedure Filltable

We suppose that in the lhs of the rule under examination there are $k-1$ distinct generics with ranges (some of the generics may have the same range) being alphabets with cardinal numbers p_0, p_1, \dots, p_k ; (the p_0 corresponds to the first generic, p_1 to the second and so on). We also suppose that the elements of the corresponding alphabets A_0, A_1, \dots, A_k , are :

$$A_0 = \{a_{00}, a_{01}, \dots, a_{0p_0}\}$$

$$A_1 = \{a_{10}, a_{11}, \dots, a_{1p_1}\}$$

$$\vdots$$

$$\vdots$$

$$A_k = \{a_{k0}, a_{k1}, \dots, a_{kp_k}\}$$

Then all the possible combinations of the sequences of simple symbols in lhs (if we exclude all the appearances of simple symbols and every other appearance of any generic, but the first one in the lhs) may be represented by the paths of the tree of the diagram 4.

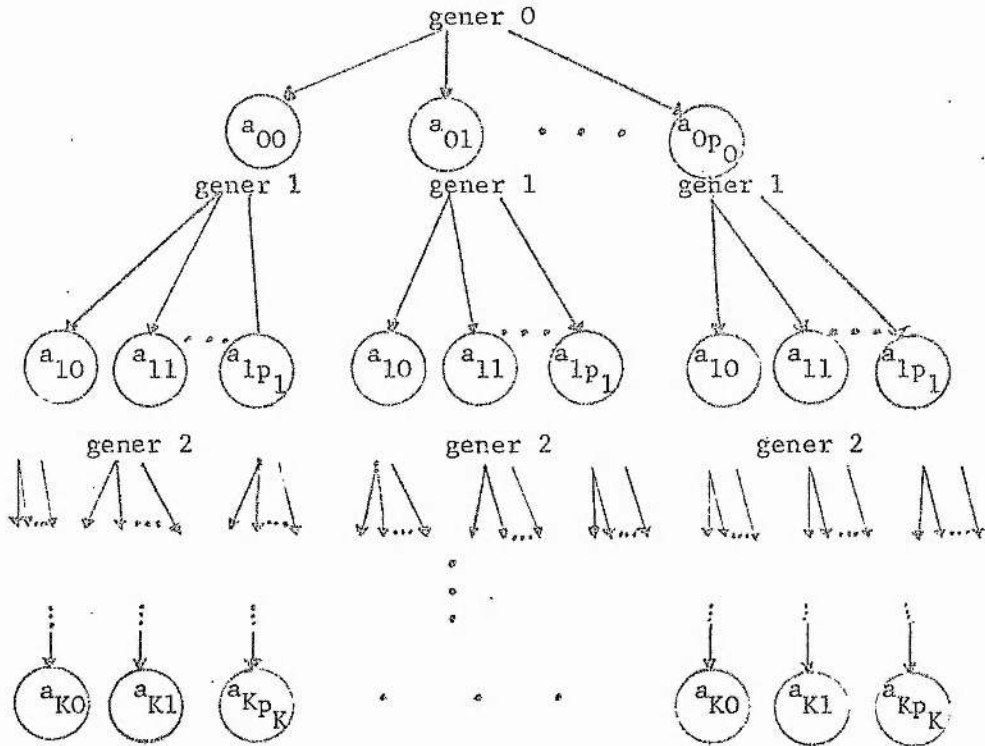


Diagram 4

Thus we see that there are $p_0 \cdot p_1 \dots p_K$ terminals in the tree, which means that the number of paths is also $p_0 \cdot p_1 \dots p_K$. As we have said we use the variable gencomb to store this product. For the representation of all these paths we use a two dimensional integer array called table, in such a way that every column of the array is a path of diagram 4. This means that the number of "lines" of the array is equal to the number of distinct generics in lhs, and the number of columns is equal to gencomb (that is $p_0 \cdot p_1 \dots p_K$).¹

1 The only difference between table and the tree of diagram 4 is that the paths are ordered differently. This however depends merely on the programming technique followed, in storing the paths in the array table.

Thus the table corresponding to the tree of diagram 4 is:

	0	1	2	3 gencomb	
0	$a_{00}a_{01}\dots a_{0p_0}$	$a_{00}a_{01}\dots a_{0p_0}$	$a_{00}a_{01}\dots a_{0p_0}$	$a_{00}a_{01}\dots a_{0p_0}$	\dots	$a_{00}a_{01}\dots a_{0p_0}$	1
1	$a_{10}a_{10}\dots a_{10}$	$a_{11}a_{11}\dots a_{11}$	$a_{12}a_{12}\dots a_{12}$	\dots	\dots	$a_{1p_1}a_{1p_1}\dots a_{1p_1}$	
2	$a_{20}a_{20}\dots a_{20}$	$a_{20}a_{20}\dots a_{20}$	$a_{20}a_{20}\dots a_{20}$	\dots	\dots	$a_{2p_2}a_{2p_2}\dots a_{2p_2}$	
.							
.							
.							
K	$a_{K0}a_{K0}\dots a_{K0}$	$a_{K0}a_{K0}\dots a_{K0}$	$a_{K0}a_{K0}\dots a_{K0}$	\dots	\dots	$a_{Kp_K}a_{Kp_K}\dots a_{Kp_K}$	

We now use the following technique. Whenever the above table is produced, if we symbolize the column 0 with $(jlast - jstart)^2$, we can specify any path (column) by using a variable jvar such that:

$$jlast - jstart \leq jvar \leq \text{gencomb} + (jlast - jstart).$$

Thus whenever in the future (that is during the generation of the application moves or even in the course of the generation of the next moves needed for the applicability search) a used generic is met in lhs, or in rhs, we can have by the pair generic, state an easy access to the cell of the array table which stores the simple symbol, into which the generic was translated in its first

- 1 The line 0 is not used in the actual array of the program as we do not symbolize any generic by 0.
- 2 jstart is the leftmost state of the last generated level of states in the tree of diagram 3. Similarly jlast is the rightmost state of the same level. As we have explained regarding the representations of the states $Q(i,j)$, we use two subscripts. Although jlast and jstart represent the second subscript we assume that they also represent the state $Q(i,j)$ because i is not modified during the examination of rule i.

appearance in lhs. This is actually very important, because when the T_m under construction is ready, the only way of memorizing the symbols into which every generic in lhs of a rule was translated, is the state of T_m . Thus we see that for every path (column) in the table, there corresponds one and only one state of T_m .

In all this we do not imply that the distinct generics in the lhs of a rule must follow one after the other. On the contrary, any number of simple symbols or already used generics may exist between two distinct generics, and this does not modify at all the use of array table, as can be seen from the tree of diagram 3.

4.8 Procedure Firstmoves

This procedure generates the first T_m moves which are similar for every M_a rule, and the moves for every inapplicable rule. By using a local array distinct it avoids the repetition of some moves already generated. This happens whenever during the applicability search the next character scanned in the tape is not the expected one and in some (indeed most) cases the head must be driven to the left at the position where the square with the marker $T_{(s)}$ is.

This is also the procedure generating the moves, of the applicability search which print the marker $T_{(s)}$, whenever the first symbol in lhs matches with the scanned character in the tape.

Finally Firstmoves calls the procedure Firstsym if the first character in lhs of the rule under examination is a simple symbol, or it calls the procedure Firstgener if that character

is a generic.

4.9 Procedure Firstsym

This procedure generates one move for each character in Alphabet, starting from cell 6 as we have explained before, in such a way, that for every character represented by a number different from the one which represents the first simple symbol in ℓhs , the "next" state of the move remains the same and the character to be typed is the character "read", that is the first in ℓhs ; for the first character in ℓhs , it generates a move and creates a next state.

This procedure also examines whether $|\ell\text{hs}| = 1$, in which case the marker $T_{(s)}$ is not used, as no more searching to the right will be made in this case by T_m . In this case, as in the corresponding one of chapter I, three cases corresponding to the lengths of ℓhs and rhs are considered. Thus a substitution move is generated for the unique character in ℓhs , by an X or a Y or the corresponding symbol of rhs .

4.10 Procedure Symbexp

This procedure is called when the ℓhs of the rule under examination is of length one, and $|\text{rhs}| > 1$ for the same rule. In this case the tape must be expanded. Thus Symbexp arranges that all the necessary moves will be generated which (for any string) will type a Y (represented by 2) on the first blank square of the tape and then they will transfer Y to the first square to the right of the one with the X already printed.

After the generation of these moves the next moves of T_m are generated by the procedure expand. So symbexp produces suitable

moves in such a way that there will be a sequence of moves followed by T_m in any case, and after the execution of the routine "expansion", T_m will continue to execute its next moves generated also by sybexp. The difficulty in the cases where T_m enters in "expansion's" execution (chapter I) is that it must always finish in the last state in which it was just before the execution of "expansion" started. For this reason there is a move generated, in all these cases (when "expansion" will be used by T_m), which types a marker $T_{(j)}$ memorising the second subscript j of T_m . This $T_{(j)}$ is printed in the square with the X. Since in this present case $|lhs| = 1$, and since we need another marker $T_{(d)}$ whenever the tape is to be expanded, sybexp arranges for the tape to be expanded by one square as we described before and thus the two markers can be typed in the two successive squares with X and Y.

Finally sybexp generates the moves which substitute the part of the tape covered by X's by the rhs of the examined rule, and also the last moves corresponding to this rule.

4.11 Procedure Firstgen

This procedure is called from Firstmoves if the first symbol in the lhs of the examined rule is a generic variable. It performs a very similar task to that of Firstsym, but it is more complicated (because of the generic), and it generates a greater number of moves. It notifies to the field used by the corresponding line of the structure Generic that the examined generic has been used and finally it calls the procedure Genexp which is similar to sybexp if $|lhs| < |rhs|$

for the rule under examination.

4.12 Procédure Nextmoves

If the lhs of the rule under examination consists of more than one element, the procedure nextmoves is called from main. This procedure continues the generation of moves necessary for the applicability search of the rule, by using a loop where each character, starting from the second, in lhs is examined; if this character is a simple one, the procedure Simplesym is called; if it is a generic which has been already used the procedure Usedgen is called; if it is a not yet used generic then the procedure Unusedgen is called.

4.13 Procédure Simplesym

Simplesym generates the usual moves for the case where all the characters from the first up to the one under examination in lhs appear successively in the string; it also generates the moves corresponding to all the other cases where the scanned character of the string is not the same with the one under examination in lhs. If the character under examination is the last in lhs, Simplesym generates the move which substitutes the corresponding character in the tape with an X or a Y or the last character of rhs. If the corresponding character of rhs is a generic, it is determined from array table to which character this array corresponds.

4.14 Procedure Usedgen

As we have said, if the first appearance of a generic in lhs stands for a character "C", then all its next appearances in lhs

or in rhs must be "translated" into the same character "C". We are interested here in the possible next appearances in lhs. Thus it is detected, from array table again, to which character the generic corresponds. In everything else Usedgen is similar to Simplesym.

4.15 Procedure Unusedgen

This is similar to Usedgen and Simplesym but it generates a much greater number of moves, as is obvious.

4.16 Procedure Caseone

Caseone is called from main when:

$$a. \quad |lhs| = |rhs|$$

or

$$b. \quad |lhs| > |rhs| \quad \text{and} \quad |lhs| > 1. \quad ^1$$

This procedure generates the necessary moves for the substitution of the first appearance of the lhs of the rule under examination, in the tape by rhs. It starts from right to left and from the penultimate characters of lhs and rhs, since the substitution move for the last character has been generated by other procedures. Thus Caseone examines the lhs and rhs character by character and generates different moves when one or both characters (that is the one in lhs and the other in rhs) are generics.

After the substitution moves have been generated, Caseone examines the lengths of lhs and rhs and if they are equal, it

1 The case $|lhs| = 1$ is examined in procedure Firstsym

generates the final moves for this rule.

If however $|lhs| > |rhs|$ there are other moves needed for the "cut-off" of the tape and the procedure Casethree is called.

4.17. Procedure Casethree

When this procedure is called, the moves which substitute the part of lhs, (equal to the length of rhs) on the tape by rhs have already been generated.

Thus Casethree generates the moves which will substitute the next to the right character of the tape by a Y and the next $|lhs| - |rhs| - 1$ characters by X's. Obviously more moves are generated if there are generics in lhs and the situation becomes more complicated.

Casethree generates also all the necessary moves which will "prepare" T_m to start the execution of subroutine "Cut-off". It also calls the procedure Aftercontract which generates the moves restoring the state which T_m had before it started the execution of "Cut-off".

4.18. Procedure Casetwo

This procedure is called from main when $|rhs| > |lhs|$ with $|lhs| > 1$, for the rule under examination.

It generates the following types of moves:

- a. The moves which are necessary for the substitution of the first appearance of lhs on the tape by X's.
- b. The moves which will arrange so that T_m will start the execution of the subroutine "expansion".

Between these moves are the ones which will print the markers $T_{(j)}$ and $T_{(d)}$ on suitable squares of the tape.

- c. The moves (or move) which will call the subroutine "expansion".
- d. The moves (or move) after the execution of expansion which will restore the last state of T_m and simultaneously substitute the left most X on the tape with the first character in lhs, or with the character corresponding to the generic if the first symbol in lhs is a generic.
- e. The moves which substitute every X from left to right by the corresponding character (or matching character) of rhs.
- f. The final moves for the rule as in Caseone.

4.19 Procedure Expand

This procedure is called from main only if at least for one rule of M_a it obtains: $|lhs| < |rhs|$. It generates all the moves necessary for the expansion of the tape which are similar to the ones of subroutine "expansion" used in chapter I.

4.20 Procedure Contract

This procedure, which is called from main only if for at least one rule it happens that $|lhs| > |rhs|$, generates the moves of subroutine "Cut-off" of chapter I.

4.21. Procedure Put, Nextrule

Procedure Put returns, whenever it is called, a character which will be the one "to be typed" from the move under generation. This character may be an X or a Y, or it may be the corresponding character of the rhs of the rule under consideration.

Nextrule is the procedure generating the moves which determine the next rule to be tested, in any case.

5. A PROGRAM PRODUCING A MARKOV ALGORITHM EQUIVALENT TO A GIVEN TURING MACHINE

This program is the direct result of an application of chapter II to a computer. It is given exactly the same input as the Turing Machine simulator program and produces output of exactly the same form as the input to the Markov Algorithm Simulator Program.

The states of the Turing Machine are represented with two subscripts, that is, in the form $Q(i,j)$. Although this is in fact more complicated, we nevertheless use this representation in order to retain a kind of uniformity in the input of all programs. Thus we will represent every state q_j of a given Turing Machine by $Q(0,j)$.

As has been seen from the construction of chapter II, a Markov Algorithm constructed so as to be equivalent to a given Turing Machine has more rules than the ones corresponding to the Turing Machine's moves. The subset of rules which will be the same for any Turing Machine will be automatically constructed by the program (by being incorporated in the program).

Finally the Markov Algorithm simulator, will be labelled, in the sense that after the application of a rule, the next rule to be tested can only be chosen from the set of rules having the appropriate marker (determining the corresponding state of the Turing Machine) in their lhs.

5.1 Data Structure

In this program we use one array structure called rules with the following fields: two character array fields with three and four elements, called lhs and rhs, respectively; two single integer fields qleft and qright; and finally two single integer fields numrule and label.

In the character array fields lhs and rhs we store the left and right hand sides of a rule when they have been generated. Thus for a "right" Turing Machine move, only two elements of the lhs field are needed and two of the rhs field; but in the case of a type zero production, that is whenever a B appears in the lhs of a rule, three elements of rhs are needed. For every "left" Turing Machine move all the elements of lhs will be used and three or all of rhs, depending on whether or not there appears a B in lhs.

In the fields qleft and qright we store the integers representing the states of the Turing Machine before and after the move (that is the previous and present markers representing these moves in the string of the Markov Algorithm).

The two last integer fields are used for ordering and attaching labels to the Algorithm.

The array "alphabet", of character type, is used for storing all the symbols used by the algorithm, including the one or two generics needed in the program. Then every character is represented by the number of the cells of alphabet that it occupies. If now the last character in alphabet is represented by the integer i , we represent a state of the Turing Machine $Q(0,K)$ by $\text{min} + K$, where $\text{min} = i+1$.

When all the Turing Machine moves have been translated to Markov Algorithm rules and stored in the fields lhs and rhs of the array structure rules, the integer array ordering is used in such a way that in its first cell an integer is stored determining the first "line" of array structure rules with the marker p (that is q_p) in its lhs field, where p is the smaller second subscript for all Turing Machine states $Q(j)$, (or $Q(0,j)$). Then the next rule with a p in its lhs follows and so on. If no other markers with the subscript p exist among the other rules, the next stored rule in the next available cell of ordering is the one with the marker q_{p+1} in its lhs and so on. Thus in this ordering we have an ordering of the rules. The integer array Firststate helps the final ordering of the rules by keeping in its i th cell an integer determining the first rule (as ordered in the field numrule of structure rules) including q_i in its lhs.

5.2 Procedures used

The procedures initone and initwo are initiallizing ones. The first sets up most structures and arrays to be used, reading from input. (There are the same ones as in the case of the Turing Machine simulator program.)

The second reads the Turing Machine moves, again from input, one by one and for every move stores the corresponding states in the fields qleft and qright of the structure rules. This procedure also calls the routine makerule which "translates" the move into an equivalent Markov Algorithm rule, and stores it in the array structure rules.

Then the main program by using the field numrule of the array structure rules and the array firststate achieves an ordering of the rules, in the array ordering. Finally the procedure Markinp produces the output which is the Markov Algorithm program and can be used as input to the Markov Algorithm simulator program.

BIBLIOGRAPHY

- J.M. Brady: "The Theory of Computer Science, A Programming Approach", Chapman and Hall, 1977.
- Martin Davis: "Computability and Unsolvability", McGraw-Hill, 1958.
- Galler and Perlis: "A View of Programming Languages", Addison-Wesley.
- Hopcroft and Ullman: "Formal Languages and their Relation to Automata", Addison-Wesley, 1969.
- David Hopkin and Barbara Moss: "Automata", MacMillan Press Ltd., 1976.
- Reino Kurki-Suonio: "A Programmers Introduction to Computability and Formal Languages", Student Litteratur, 1971.
- Zohar Manna: "Mathematical Theory of Computation", McGraw-Hill, 1974.
- A.A. Markov: "Theory of Algorithms (Theoriya Algorifmor)", Academy of Sciences of the USSR.
- Marvin Minsky: "Computation, Finite and Infinite Machines", Prentice-Hall International, 1972.

R.S. Nelson: "Introduction to Automata", John Wiley and
Sons, Inc., 1968.

Arto Salomaa: "Formal Languages", Academic Press, 1973.